2014

Biogeography-Based Optimization for Combinatorial Problems and Complex Systems

Dawei Du
Cleveland State University

BIOGEOGRAPHY-BASED OPTIMIZATION FOR COMBINATORIAL PROBLEMS AND COMPLEX SYSTEMS

DAWEI DU

Master of Science in Electrical Engineering

Cleveland State University

August, 2009

This dissertation is submitted in partial fulfillment of requirements for the degree

DOCTOR OF ENGINEERING

at the

CLEVELAND STATE UNIVERSITY

April, 2014

We hereby approve the dissertation
of
Dawei Du

Candidate for the Doctor of Engineering degree.
This dissertation has been approved for the Department of

---

and CLEVELAND STATE UNIVERSITY
College of Graduate Studies by

Dr. Dan Simon, Dissertation Committee Chairperson – Department & Date

---

Dr. Lili Dong, Dissertation Committee Member – Department & Date

---

Dr. Wenbing Zhao, Dissertation Committee Member – Department & Date

---

Dr. Yongjian Fu, Dissertation Committee Member – Department & Date

---

Dr. Yuping Wu, Dissertation Committee Member – Department & Date

---

Student's Date of Defense

The student has fulfilled all requirements for the Doctor of Engineering degree
Dr. Dan Simon, Doctoral Program Director

ACKNOWLEDGMENTS

I would like to thank the following people: Dr. Dan Simon, for all his diligent guidance as my supervisor, and his unselfish help in all aspects of my research; Dr. Jeffrey Abell, for his valuable advice and ideas; my other committee members, Dr. Lili Dong, Dr. Wenbing Zhao, Dr. Yuping Wu, and Dr. Yongjian Fu, for their time and suggestions; Mehmet Ergezer, Richard Rarick, Oliver Tiber, Steve Szatmary, George Thomas, Berney Montavon, and David Sadey, for their patience in giving me all the help I needed; and my wife Yuanchao Lu and my entire family, for their support in my life.

BIOGEOGRAPHY-BASED OPTIMIZATION FOR COMBINATORIAL PROBLEMS AND COMPLEX SYSTEMS

DAWEI DU

ABSTRACT

Biogeography-based optimization (BBO) is a heuristic evolutionary algorithm that has shown good performance on many problems. In this dissertation, three problems are researched for BBO: convergence speed and optimal solution convergence of BBO, BBO application to combinatorial problems, and BBO application to complex systems. The first problem is to analyze BBO from two perspectives: how the components of BBO affect its convergence speed; and the reason that BBO converges to the optimal solution. For the first perspective, which is convergence speed, we analyze the two essential components of BBO – population construction and information sharing. For the second perspective, a mathematical BBO model is built to theoretically prove why BBO is capable of reaching the global optimum for any problem. In the second problem addressed by the dissertation, BBO is applied to combinatorial problems. Our research includes the study of migration, local search, population initialization, and greedy methods for combinatorial problems. We conduct a series of simulations based on four benchmarks, the sizes of which vary from small to extra large. The simulation results indicate that when combined with other techniques, the performance of BBO can be significantly improved. Also, a BBO graphical user interface (GUI) is created for

combinatorial problems, which is an intuitive way to experiment with BBO algorithms, including hybrid BBO algorithms. The third and final problem addressed in this dissertation is the optimization of complex systems. We invent a new algorithm for complex system optimization based on BBO, which is called BBO/complex. Four real world problems are used to test BBO/Complex and compare with other complex system optimization algorithms, and we obtain encouraging results from BBO/Complex. Then, a Markov model is created for BBO/Complex. Simulation results are provided to confirm the model.

TABLE OF CONTENTS

Page

NOMENCLATURE

D_{ghab}: partial distance between island a in archipelago g and island b in archipelago h

$J(s)$: set of islands which contain the same feature as the s-th feature in island v

k: total number of features in each island

m: number of objectives

N: total number of possible solutions in the entire system

N_i: population size of subsystem i

n: search space size

n_f: total number of candidate features in the s-th position

$n_{feature}$: number of features in the feature pool

n_i: cardinality of search space in subsystem i

$n_{in\text{-}common}$: number of features common to individual k and the optimal individual

$num(i, j)$: number of occurrences of feature j in individual i

P: transition matrix

P_{ss}: steady state transition matrix

p_c: probability of recombination

p_i: migration probability

p_m: mutation rate

$p_{mutation}$: mutation probability

p_o: probability that at least one individual becomes the optimal solution after one recombination

Pr(u=v): probability that individual v becomes individual u

Pr(u_{si}): probability that u is selected for immigration

Pr_g(u=v): probability that individual u becomes individual v in the g-th generation

R_i: rank of the i-th island

T_i: total number of possible population vectors in subsystem i

t: total number of SIV types

U_i: mutation matrix

V_i: number of constraint violations of the i-th island

v: population vector

σ_{ilmj}: distance between island l in subsystem i and island j in subsystem m

λ: immigration rate

μ: emigration rate

ACRONYMS

ACO: ant colony optimization

AIAA: American Institute of Aeronautics and Astronautics

BBO: biogeography-based optimization

BBO/CO: BBO with circular opposition

BBO/TSP: BBO for TSPs

BBO/Complex: BBO for complex systems

CO: collaborative optimization

DE: differential evolution

EPA: extreme point algorithms

ES: evolutionary strategy

GA: genetic algorithm

GA/MSX: genetic algorithm with multistep crossover

HSI: habitat suitability index

I/O: input/output

IDF: individual discipline feasible

MDA: multidisciplinary analysis

MDO: multi-disciplinary design optimization

MDF: multidisciplinary feasible

MOGA: multi-objective genetic algorithm

NDRS: non-dominated ranking system

NNA: nearest neighbor algorithm

NSGA: non-dominated sorting genetic algorithm

OX2: order-based crossover operator

PBIL: probability-based incremental learning

PDS: partial distance strategy

PSO: particle swarm optimization

FSLC: fast similarity level calculation

SA: simulated annealing

SGA: stud genetic algorithm

SIV: suitability index variable

SL: similarity level

TSP: traveling salesman problem

TSPBMA: biogeography migration algorithm for traveling salesman problem

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER I

INTRODUCTION

1.1 Biogeography-Based Optimization

With the advance of today's technology, simple systems cannot satisfy the needs of industry. Complex systems have become the mainstream. Control and optimization are more complicated and challenging as system complexity increases. Sophisticated algorithms designed for special types of problems have been invented, requiring a full understanding of these problems. But if we turn to heuristic algorithms, it is not necessary to completely understand the system before applying them for control or optimization. In contrast with other algorithms which are designed for special types of problems, heuristic algorithms can easily adapt to almost any type of problem with only minor changes. The main drawback of the heuristic algorithm is that it needs long computation time before achieving desirable results. But with powerful computers, this drawback is tolerable.

Biogeography-based optimization (BBO) is an algorithm which was introduced in 2008 [1]. This algorithm is inspired by the distribution of species over time and area. The environment of BBO is an archipelago which consists of islands, where each island includes many species (features). Each feature is called a suitability index variable (SIV). Each island is considered as a potential solution to an optimization problem. The performance of each solution is evaluated by the problem's cost function, and we use the habitat suitability index (HSI) to indicate the level of performance. The method to share features between islands is called migration and the method to randomly modify an island is called mutation. These two methods describe the evolution of the population in BBO.

The basic procedure of the BBO algorithm is as follows:

1. Define the mutation probability, and elitism parameter. Mutation and elitism are the same as in genetic algorithms (GAs) [2].
2. Initialize the population.
3. Calculate the immigration rate and emigration rate for each island. Good solutions have high emigration rates and low immigration rates. Bad solutions have low emigration rates and high immigration rates.
4. Probabilistically choose the immigrating islands based on the immigration rates. Use roulette wheel selection [3] based on the emigration rates to select the emigrating islands.
5. Migrate randomly selected SIVs based on the selected islands in the previous step.
6. Probabilistically perform mutation based on the mutation probability for each island.
7. Calculate the fitness of each individual island.

8. If the termination criterion is met, terminate; otherwise, go to step 3 for the next generation.

The original BBO algorithm shows good potential when compared over 14 benchmark problems with seven well-know competitors – ant colony optimization (ACO), differential evolution (DE), evolutionary strategy (ES), GA, probability-based incremental learning (PBIL), particle swarm optimization (PSO), and stud genetic algorithm (SGA) [1]. Due to its performance, it is widely used in many areas, such as power control [4], fuzzy robot controller tuning [5], and traveling salesman problem (TSP) [6]. Also, a Markov model [7] and dynamic system model [8] have been derived for BBO, which can predict the performance of BBO theoretically before applying it to real world problems. They are useful methods to analyze the performance of BBO and also provide a solid proof why BBO obtains such good performance.

Although BBO achieves outstanding results in benchmark tests, it still has room to improve. Most heuristic algorithms are considered as a framework, or family of algorithms. Taking GAs as an example, many GAs are invented for different purposes. Examples include: non-dominated sorting genetic algorithm (NSGA) [9], genetic algorithm with multistep crossover (GA/MSX) [10], etc. These algorithms all belong to the GA family, but their details are different. Most of the details in a heuristic algorithm can be modified or replaced for different types of problems to gain maximum performance.

In this dissertation, we perform an analysis of how to increase the efficiency of a heuristic algorithm. The efficiency metric is the convergence speed. BBO is used as an

example to demonstrate the analysis. It can also be considered as a guideline for how to create a hybrid BBO with better efficiency.

Combinatorial problems are NP-hard problems [11], and their large search spaces make them incompatible with traditional mathematical methods. This makes them a perfect benchmark for heuristic algorithms. For the demonstration and simulation purposes of this dissertation, the traveling salesman problem (TSP) is used as the prototypical example of a combinatorial problem. For a 100-city TSP, the total number of candidate solutions is $100! = 9.3326 \times 10^{157}$. Using exhaustive search methods is a dead end for this type of problem. BBO has the potential to be a powerful tool for combinatorial problems. In this dissertation, we create hybrid BBO algorithms with high efficiency for combinatorial problems.

The final contribution of this dissertation is to apply BBO to complex systems, which consist of multiple interacting subsystems. Each of the subsystems has multiple objectives and multiple constraints. The reason for applying BBO to complex systems is that a complex system includes three factors which cannot be easily addressed and solved by traditional methods: multi-systems, multi-objectives, and multi-constraints. Since complex systems are commonly used in today's industry, providing a solution method for complex systems can be a significant contribution to industry. Also, these three factors are difficult even for heuristic algorithms [9]. As a heuristic algorithm, BBO faces the same challenge.

1.2 Literature Review

BBO has proven its performance based on comparisons with other algorithms in a series of benchmark tests [1]. These tests can be roughly considered as efficiency tests based on the convergence time and final results. In [12], [13], [14], [15], and [16], hybrid BBO algorithms are introduced for different types of problems and circumstances. The simulation results from these papers show performance improvement compared to the original algorithm in certain areas.

1.2.1 Combinatorial Problems

Combinatorial problems are not new to heuristic algorithms. They are considered as standard benchmarks for heuristic algorithms. Combinatorial problems represent a special category of problems. Inside this category, there are many subcategories. Some of them have significant effects in our daily life. For example, the vehicle routing problem, the knapsack problem, the TSP, etc.

Vehicle routing problems were first proposed in 1959 [17]. For this type of problem, the aim is to design the optimal route for picking up or delivering people or goods from one or several locations to a number of scattered locations with certain constraints. Vehicle routing problems are a common type of combinatorial problem, and many real world problems, like bus routing [18], mail delivery [19], etc., belong to this category.

The knapsack problem is another type of combinatorial problem. It can be traced back to 1897 [20]. The description of this problem is: when given a set of objects which have different weights and values, choose some objects from this set to maximize the total value but still be under the weight limit. It also appears in real world applications such as selection of capital investment [21].

TSP, a famous combinatorial problem, is an ancient problem whose origins have been lost in the mists of history. The TSP was first formulated as a mathematical problem by Karl Menger in 1930 [22]. There are three major reasons that the TSP has become a standard benchmark for heuristic algorithms. First, the TSP is an easily stated problem and is similar to many practical problems, such as sensor selection [23], the mailman problem [24], robotic path planning [5], and many others. Second, the TSP can easily be modified to become a multi-objective problem [25], and solving multi-objective problems is a practical challenge in many areas of engineering and industry. Third, the optimal TSP solution is extremely hard to find using analytical methods. Even using numerical methods, it is still quite a challenge.

In [26], BBO has been applied to TSPs. The new algorithm is called the biogeography migration algorithm for traveling salesman problem (TSPBMA), which is a specially modified version of BBO for combinatorial problems which achieves good results. In [27], BBO with circular opposition (BBO/CO) was introduced as a modified version of BBO which achieved promising results for 16 TSP benchmarks. Two techniques are implemented to create BBO/CO: circular opposition and combinatorial BBO migration, which is also called the simple version of inver-over crossover that will be introduced in the following sections of this dissertation. Although specially designed

BBOs were invented in those papers, they only discussed modification in the migration component. But in this dissertation, besides the migration component, the discussion will be extended to new areas in BBO including the population construction, the local search optimization, and the greedy method.

1.2.2 Complex Systems

The material in this section is based on [28], which is one of the dissertation author's publications, and which is used here with permission. Complex systems have become an important topic. In [29], we read that a complex system has the following properties: 1) a complex system contains a large number of elements; 2) the elements have interactions with each other; 3) the interactions are rich; 4) the interactions contain certain complex characteristics such as nonlinearity. In [30], a complex system is defined as "[a]n assembly of interacting members that is difficult to understand as a whole." Complex systems can have various structures, as long as they satisfy the above definitions.

The mathematical description of a system comprises equations and inequalities that include the definitions of variables, the ranges of variables, and the connections between variables. Optimizing a system is equivalent to mathematically defining the system, and then finding the feasible solutions that (approximately) optimize the objective functions. But when the order of the equations or inequalities is relatively large, or those equations or inequalities are highly nonlinear, the solutions must be obtained numerically rather than analytically [31]. Unfortunately, most complex systems include interacting subsystems that are either continuous or NP-hard, and thus contain a huge

number of possible solutions. The inclusion of subsystems in complex systems adds even more complexity than that involved in a single system.

For example, a complex system can have a multilevel structure, such as a decentralized planning problem with multiple executors in a hierarchical organization. The simplest case of a multilevel problem is the bilevel problem [32]. The description of a bilevel problem is as follows.

$$\begin{aligned} &\min_{x,y} H(x,y) \\ &\text{subject to } F(x,y) \le 0 \end{aligned} \tag{1.1}$$

Equation (1.1) describes an upper level problem. In contrast, a lower level problem is described as

$$\begin{aligned} &\min_{y} h(x,y) \\ &\text{subject to } f(x,y) \le 0 \end{aligned} \tag{1.2}$$

In Equation (1.2), for each value of x, there exist a solution, y. Variable x is called the upper level variable, and y is called the lower level variable. $F(x, y)$ is the upper level constraint, and $f(x,y)$ is the lower level constraint. The bilevel problem is a special case of a multilevel problem. When a problem has multiple levels in a hierarchical organization and also has connections as shown in Equation (1.1) and (1.2), it is called a multilevel problem.

Many real world applications are typical multilevel problems. One example is the aircraft design problem [33], which is extremely complicated and involves thousands of components. Network design [34] is another multilevel problem, whose goal is to optimize the balancing of transportation, construction costs, and maintenance costs of a

network. It is similar to the aircraft design problem: large size, large number of components, and extreme difficulty for optimization. Besides these two problems, coordination of multidivisional firms [35], and electric utility planning [36] are also considered multilevel problems. Many algorithms have been invented to solve these types of problems, such as extreme point algorithms (EPA) [37] and collaborative optimization (CO) [38], both of which belong to the multi-disciplinary design optimization (MDO) category.

In the 1970s and 1980s, computer aided design became a mature approach for aircraft design, including economic factors, manufacturability, reliability, etc. Aircraft design was the initial motivation of MDO [39]. With thousands of parts and parameters in airplane design, MDO provided a revolution in the aircraft industry. In 1989, the American Institute of Aeronautics and Astronautics (AIAA) established the technical committee on MDO [39].

As mentioned above, MDO is a class of optimization methods. Numerous algorithms belong to this class, such as: multidisciplinary feasible (MDF), which is the most popular MDO algorithm [40]; individual discipline feasible (IDF), which does not require system decomposition [41]; and CO, which is effective for many complex systems, and which has been widely adopted in industry [38].

Traditional MDO algorithms are frameworks that provide basic conceptual structures without specifying the detailed underlying algorithms. In [42], the definition of MDO is given as follows: “an MDO method for a given problem consists of an MDO formulation and an optimization algorithm.” The particular optimization algorithm is usually chosen based on the specific problem or the user’s preference. Different MDO

methods can share the same underlying optimization algorithm. Conversely, the same MDO method can be implemented with different underlying optimization algorithms. Therefore, the major difference between MDO algorithms is the MDO formulation, or in other words, the structure of the method.

The most popular MDO algorithms include MDF, CO, and IDF. MDF is perhaps the most well known MDO algorithm. It is often considered the standard solution method for multidisciplinary problems. The structure of a typical MDF algorithm is shown in Figure 1. The top level of MDF is system optimization. The second level is called multidisciplinary analysis (MDA), which passes coupled variables among subsystems to obtain feasible solutions at the subsystem level after a certain number of iterations. After reaching the iteration limit, the second level passes its solution to the first level, and this completes one optimization cycle. The iteration cycle limit is usually defined by the user. The structure of MDF enables it to be a very competitive optimization method when the subsystems are highly coupled.

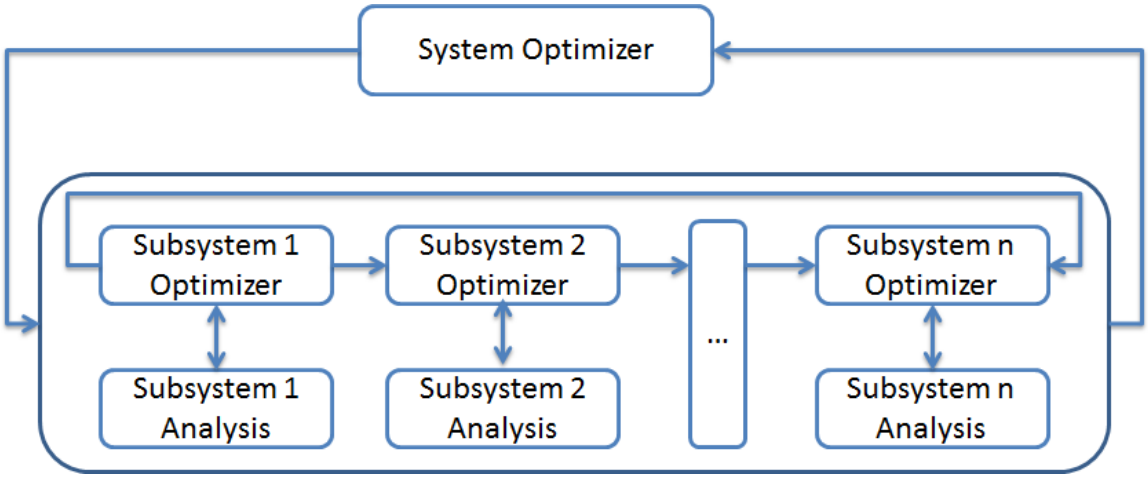

Figure 1: Multidisciplinary feasible (MDF) formulation

CO is another typical MDO algorithm, and has a bilevel structure which is shown in Figure 2. The first level is the system optimizer, which optimizes the feedback from the subsystem optimizers. The second level is the combination of the subsystem optimizers, which optimize each subsystem. Unlike MDF, the subsystem optimizations in CO are independent from each other, which means that CO puts more focus on subsystem optimization, which is advantageous for systems with extremely complex subsystems that are loosely coupled.

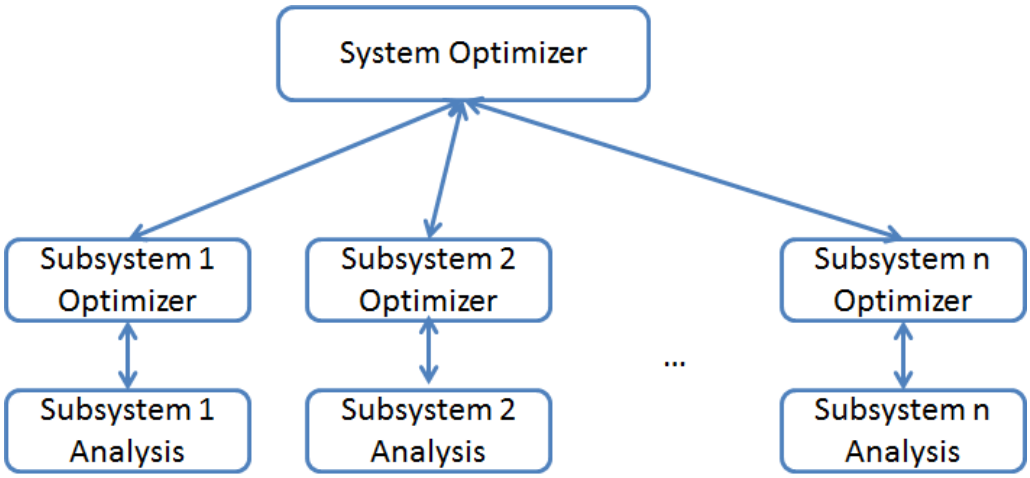

Figure 2: Collaborative optimization (CO)

IDF is an all-in-one MDO algorithm. The most significant benefit of IDF is that it can optimize all of the subsystems together without subsystem optimizations. For most MDO algorithms, decomposition of the system is necessary. But unlike CO, IDF does not require subsystem optimization. It treats subsystems more like objective functions. As long as we have the objectives and constraints for each subsystem, IDF can be implemented. As we see from the structure of IDF in Figure 3, IDF includes subsystem

analysis but not subsystem optimizers, which makes it an all-in-one algorithm. Optimization only operates at the global system level.

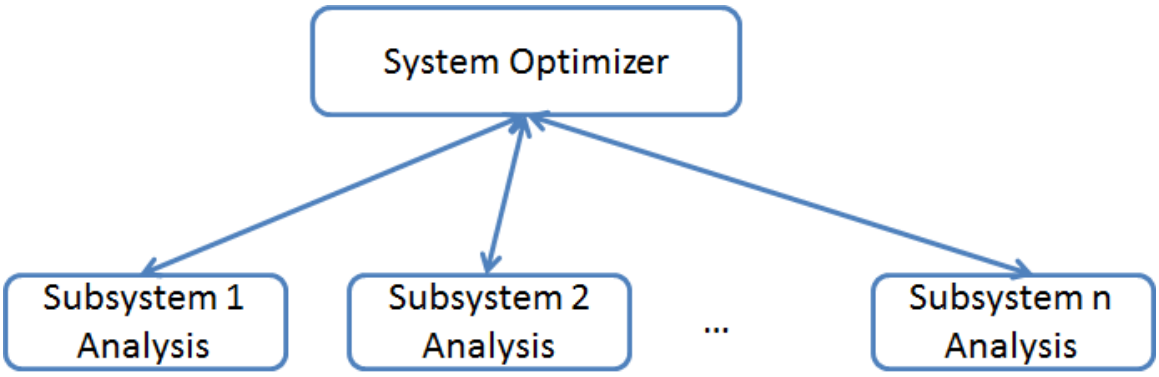

Figure 3: Individual discipline feasible (IDF) formulation

A new BBO algorithm for complex systems will be introduced in Chapter 4. Since MDF, IDF and CO are well-established algorithms in the MDO category, we will compare this BBO algorithm with those popular algorithms on four real world complex systems to reveal its potential.

1.3 Dissertation Organization

Chapter 2 comprises the first original contribution of this dissertation, where we introduce the efficiency analysis and convergence analysis for heuristic algorithms. Heuristic algorithms are usually time consuming. An efficiency analysis is used to compare the performances of algorithms. The most efficient algorithm should achieve the optimal solution with the shortest computation time. For the same algorithm, different

setups can result in very different performances. The first task of Chapter 2 is to determine the performance of algorithms under different population initialization methods, migration methods and mutation methods. The probability of convergence to the global optimal solution is always a concern when implementing a heuristic algorithm. Usually users think there is no guarantee that heuristic algorithms will eventually obtain the optimum, which results in heuristic algorithms being labeled as unreliable algorithms. But is that a true accusation? The second task of Chapter 2 is to conduct an analysis of this question.

Chapter 3 comprises the second original contribution of this dissertation, where we apply BBO to combinatorial problems. As we know, the original BBO is designed for problems with a single objective, and no constraints. So it is not originally designed for combinatorial problems. In Chapter 3, multiple modifications are applied to BBO. TSPs are used as benchmarks for performance tests. We build a BBO GUI based on Matlab® which provides a BBO framework for TSPs. There are 100 TSPs from TSPLib [43] as the default benchmark problems, and users are encouraged to implement their own algorithms and add their own benchmarks using this GUI.

Chapter 4 comprises the third original contribution of this dissertation, where the solution method for complex systems using BBO is introduced. As we know, a complex system consists of multi-subsystems, and subsystems share similar objectives and constraints. Despite the complex structure of such systems, they are common in today's industry. In Chapter 4, BBO is applied to complex systems for system optimization. Also, a Markov model is built for BBO/Complex, and simulation is provided to confirm this mathematical model.

In the last chapter, we conclude the dissertation and propose future work and directions for the next steps in research.

CHAPTER II

EFFICIENCY ANALYSIS FOR HEURISTIC ALGORITHMS

The material in this chapter is partially based on [6], which is one of the author's publications. It is used with permission. BBO belongs to the category of heuristic algorithms, which are a good complement to traditional optimization methods, especially for large, complex systems. But there are some concerns about typical heuristic algorithms, such as: 1) heuristic algorithms usually have long computation time; 2) there is no guarantee of finding the global optimum. Since BBO is a heuristic algorithm, it inevitably inherits those concerns. In this chapter, we will provide an analysis of BBO based on these major concerns.

2.1 Analysis of Performance Efficiency and Computational Speed

BBO mimics nature and can be considered an evolutionary process. Even though BBO evolution is much faster than that in nature, it still involves many individuals and needs to perform crossover and mutation for the population. Compared with the

traditional ways we solve problems, it is a slower path to the solution. Is there a way to speed up heuristic algorithms?

First, we need to analyze the reasons that heuristic algorithms are computationally intensive. A typical heuristic algorithm consists of four components:

1. Initial population construction
2. Cost calculation
3. Recombination
4. Mutation

Any of these four steps may be the source of significant computational time. Since cost calculation is problem dependent, we only analyze the remaining three components.

2.1.1 Initial Population Construction

A heuristic algorithm needs an initial population of candidate solutions, but this population construction only happens in the first generation. In the following generations, the population is updated by recombination and mutation, and this updated population is used in the next generation.

Most algorithms randomly create an initial population. This method can simplify the optimization algorithm, especially for problems with complex structure and many tuning parameters. Although population initialization is only performed once in the algorithm, it still can cause inefficiency.

First, random initialization is not an efficient method to create a population. As always, a good starting point is half the way to success. For most problems, no matter how complex they are, we usually have at least some problem specific background knowledge.

For example, suppose we try to find the optimal five features for some problem. The basic setup of BBO for this problem might be as follows: population size is 2; number of features in each individual is 5; crossover probability is 0.5; mutation probability is 0.01; and the size of the feature pool (search space) is 30.

Assume the optimal features are feature 1, feature 2, feature 3, feature 4 and feature 5. Suppose the order of features in the individual does not affect the overall performance. The optimal solution and the two individuals in the initial population might be created as shown in Figure 4. This example will be continued in the following section.

Optimal Solution	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 5**
Individual 1	**Feature 1**	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 10**
Individual 2	**Feature 5**	**Feature 11**	**Feature 12**	**Feature 13**	**Feature 14**

Figure 4: Optimal solution, individual 1, and individual 2

2.1.2 Mutation

Suppose mutation is the only method in the previous example to create individuals for the next generation. Assuming that we have an individual *k*, the probability of obtaining the optimal individual is calculated as follows.

$$p_{optimal} = \left(1 - p_{mutation} + p_{mutation}\frac{1}{n_{feature}}\right)\left(p_{mutation}\frac{1}{n_{feature}}\right)^{n_{not-in-common}} \quad (2.1)$$

$n_{feature}$: Number of features in the feature pool.

$n_{in\text{-}common}$: Number of features common to individual *k* and the optimal individual.

$n_{not\text{-}in\text{-}common}$: $n_{not\text{-}in\text{-}common} = n_{feature} - n_{in\text{-}common}$.

$p_{mutation}$: Mutation probability.

Applying Equation (2.1) to our example in Figure 4, the probability that individual 1 mutates to the optimal individual is 3.33×10^{-4}. The probability that individual 2 mutates to the optimal individual is 1.22×10^{-14}.

Thus individual 1 has a much better chance to be mutated to the optimal solution. Also, it is easy to see that the similarity level between individual 1 and the optimal solution is much higher than the similarity level between individual 2 and the optimal solution. This example shows how better population initialization achieves better efficiency.

The role of mutation is to introduce new information to the population. As the population evolves, all the individuals tend to cluster near locally optimal solutions and so the population as a whole lacks diversity. Since the probability of mutation is low, its role at the beginning of the heuristic algorithm is not critical. But closer to the end of the simulation, it becomes the only way to introduce new features required to achieve the globally optimal solution.

Can we improve the efficiency of mutation? Here, we use the same example as in Figure 4 to calculate the probability of mutating individual 1 to the optimal solution after one generation. The result is shown in Figure 5.

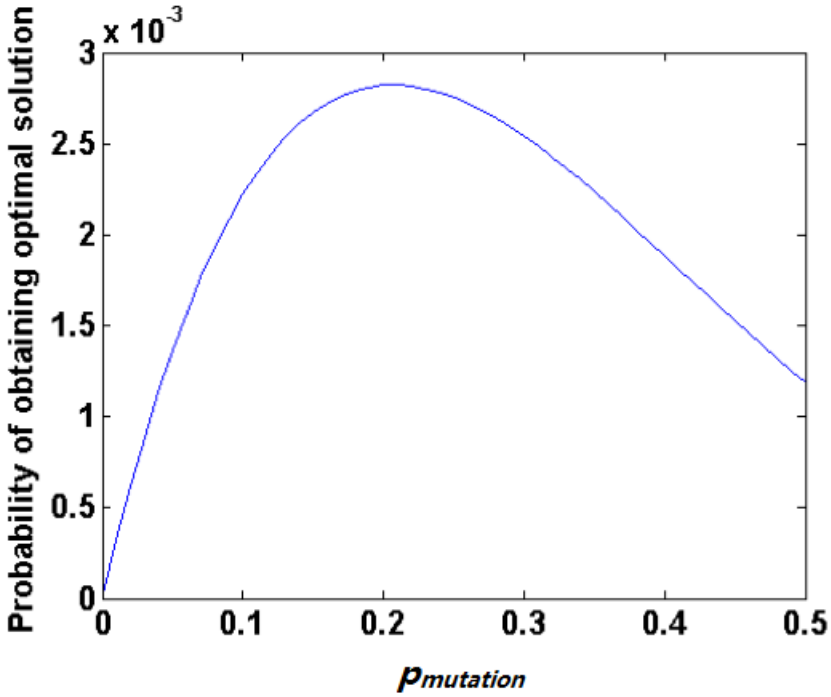

Figure 5: The probability of obtaining the optimal solution with different mutation rates after one generation

The best probability we obtain is 2.80×10^{-3} when $p_{mutation}$ is 0.21. It is 8.48 times better than when the mutation rate is 0.05. When we have a better understanding of the problem, it is easy to maximize the efficiency of a heuristic algorithm by tuning parameters in different phases. Even for mutation, different setups result in dramatic differences.

2.1.3 Recombination

The previous section only analyzed mutation. But for most heuristic algorithms, recombination is a more efficient way to create new individuals. So in this section, we analyze the probability of achieving the optimal solution based solely on crossover for different population initialization methods.

The recombination procedure is as follows:

1. Determine if individual k_1 will participate in recombination based on the calculated recombination probability. If yes, go to step 2; otherwise, check the next individual.
2. Probabilistically choose an individual to share its features based on roulette wheel selection. This individual is called individual k_2.
3. Randomly choose some features from individual k_2 to replace features in individual k_1.

Now, we create two populations for comparison purposes. The first one is randomly created, and the second one is created based on our knowledge of the problem, which is similar to creating the population in a TSP problem with the nearest neighbor strategy

(NNA) [44]. There is no guarantee that the individuals that are specially created will be closer to the optimal solution. But there should be a better chance.

Suppose we have the same optimal solution as in the previous example in Figure 4. In this example, we compare two different populations to analyze the importance of the initial population for recombination. Figure 6 shows the optimal solution, along with two possible populations, each population containing two individuals.

Optimal Solution	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5
Population 1	Feature 22	Feature 5	Feature 13	Feature 16	Feature 11
	Feature 2	Feature 10	Feature 4	Feature 5	Feature 25
Population 2	Feature 1	Feature 2	Feature 3	Feature 10	Feature 5
	Feature 1	Feature 2	Feature 20	Feature 4	Feature 6

Figure 6: Optimal solution, population 1, and population 2

Population 1 is randomly initialized, and it only contains three features from the optimal solution in the population. Population 2 is initialized by manual intervention, or expert knowledge, based on problem specific knowledge. Although no optimal individual exists in either population, population 2 contains all the necessary features to obtain the optimal solution.

After one recombination, what is the probability of obtaining at least one optimal individual? Assume we have m individuals, and each individual contains n features. Each individual has the same probability to be selected for recombination.

- p_c: The probability of recombination.
- *num(i, j)*: Number of occurrences of feature j in individual i.
- p_o: The probability that at least one individual becomes the optimal solution after one recombination.

An algorithm that calculates the probability that individual i becomes the optimal individual after one recombination is shown as follows:

1. Set $p_{o,k}$ =1, l=1, k=1
2. Determine if the l-th feature in individual i is contained in the optimal solution. We call the l-th feature in individual i feature b. If yes, go to step 3; else, go to step 4.
3. $p_{o,k} = p_{o,k}\left(1 - p_c + p_c \frac{\sum_{i=1}^{m} num(i,b)}{m \times n}\right)$. Go to step 5.
4. $p_{o,k} = p_{o,k} p_c \frac{\sum_{i=1}^{m} num(i,b)}{m \times n}$. Go to step 5.
5. If $l < n$, then $l = l + 1$, and go to step 2. Else if $k < m$, then $k = k + 1$ and $l = 1$, and go to step 2; otherwise, terminate.

The probability of obtaining at least one optimal individual in the population after performing recombination once on each individual is

$$p_o = 1 - \prod_{k=1}^{m} (1 - p_{o,k}) \tag{2.2}$$

As we show in Figure 6, population 1 is randomly initialized, and population 2 is constructed using problem specific knowledge. All individuals in population 2 are close to the optimal solution. When we set p_c = 0.5, the probability of obtaining at least one optimal individual after one recombination for population 1 is 0; but for population 2, it is 0.59%.

For the example in Figure 6, the population only consists of two individuals. For real world applications, the population size is much larger, usually over 50. In the following example in Figure 7, we increase the population size to 100. But based on problem specific background knowledge, we can narrow the feature pool size to 7 instead of 30. For population 1, its feature pool contains feature 1 to feature 7. But for population 2, its feature pool contains feature 1 to feature 30.

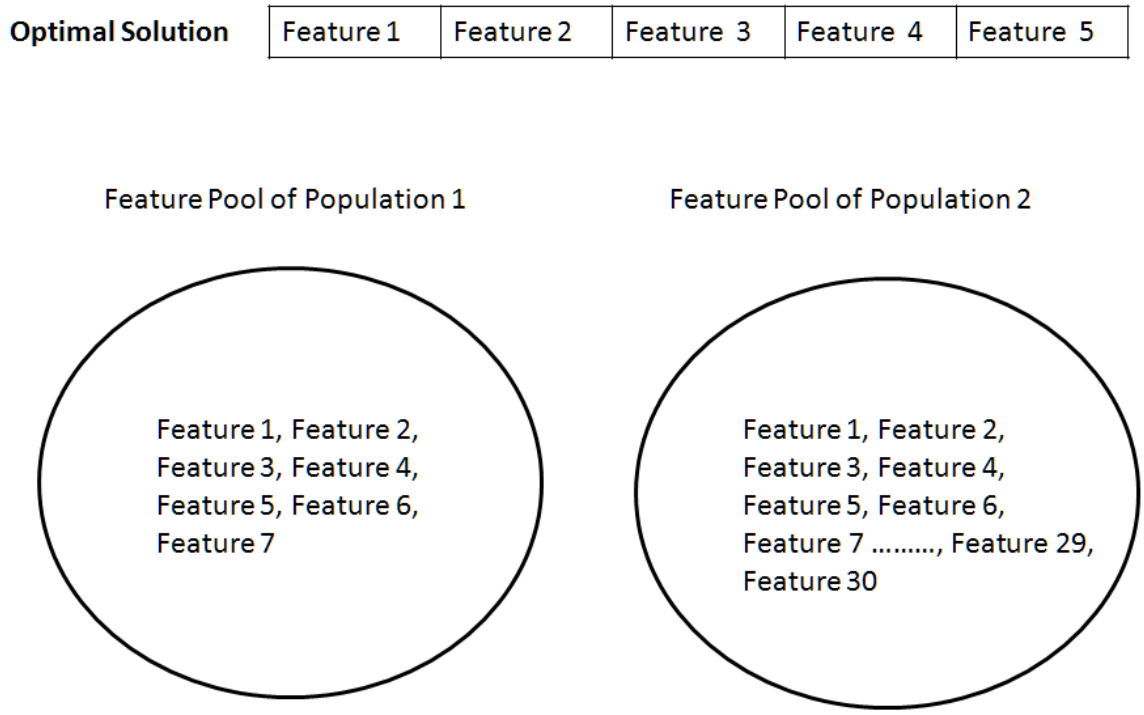

Figure 7: Optimal solution, features of population 1, and features of population 2

Let $p_c = 0.5$ as before. Then we calculate the probability of obtaining at least one optimal individual in the population after performing recombination once on each individual. For population 1, the probability is 54.85%. But for population 2, it is only 0.97%. Thus, it is more likely that a heuristic algorithm starting with population 1 will outperform the same algorithm starting with population 2 when all else is equal. It also means that population initialization can play a significant role in increasing the efficiency of a heuristic algorithm.

2.1.4 Information Sharing in TSPs

Information sharing is a key technique in heuristic algorithms. In most heuristic algorithms, we call it recombination, or crossover. Usually, there are two ways to create new individuals: one is by combining information from multiple individuals to create a new individual (crossover), and the other is to mutate an individual to obtain a new one (mutation). Mutation rates are fairly low, because high mutation rates may cause damage to the performance of the population. Statistically speaking, most of the new information it introduces is not useful. The best time for mutation is when the population converges to local optima, and new information is needed in the population. So mutation is not a rapid way to improve the overall quality of the population, and that is why crossover usually plays the key role in heuristic algorithms.

Heuristic algorithms are generally time consuming. In order to build a faster heuristic algorithm, we need to improve its efficiency in all aspects. If we make crossover more efficient, it may increase efficiency, especially for large problems. Is there room to improve the efficiency of crossover? The answer is YES, but we still need problem specific background knowledge. The flexibility of heuristic algorithms is beneficial, but we may see severe efficiency issues if we do not use using background knowledge for the algorithm design.

In the following example, we construct a closed 6-city traveling salesman problem (TSP) which involves Las Vegas, San Diego, Phoenix, Chicago, Cincinnati, and Atlanta. The locations of the cities are shown in Figure 8.

Figure 8: 6-City Problem in TSP

Figure 9 and Figure 10 show the worst and best scenario for this 6-city TSP. The worst scenario of the trip is Chicago to Las Vegas to Atlanta to Phoenix to Cincinnati to San Diego to Chicago. The best scenario of the trip is Chicago to Cincinnati to Atlanta to Phoenix to San Diego to Las Vegas to Chicago.

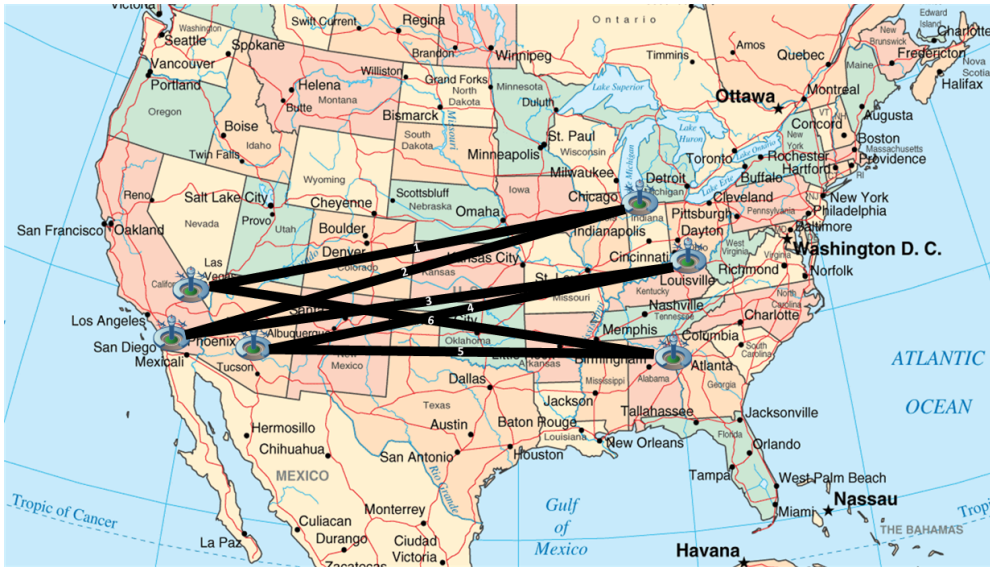

Figure 9: Worst Scenario in TSP

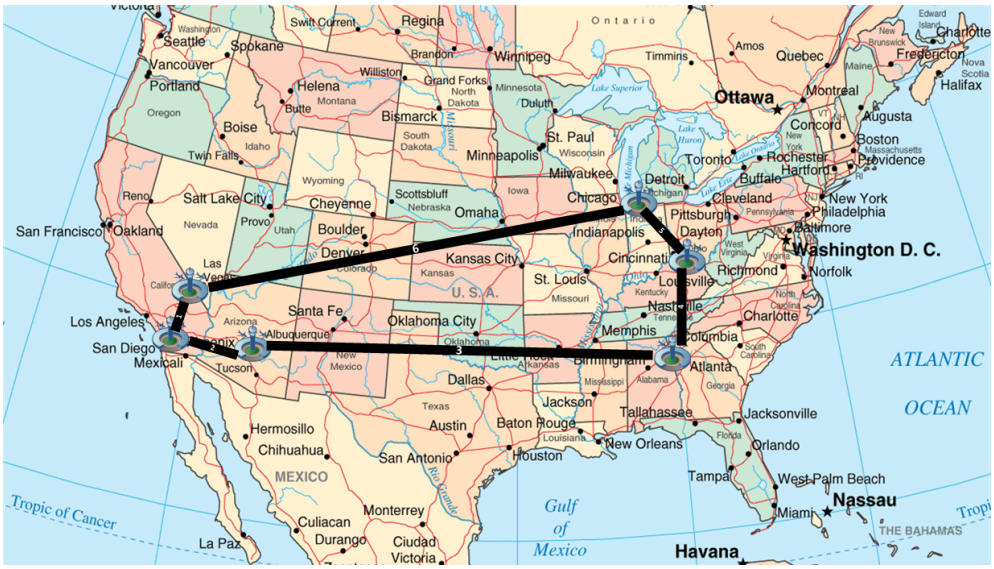

Figure 10: Best Scenario in TSP

Assume we have a population that contains two individuals, which are the best and worst scenario. The two individuals are shown in Figure 11.

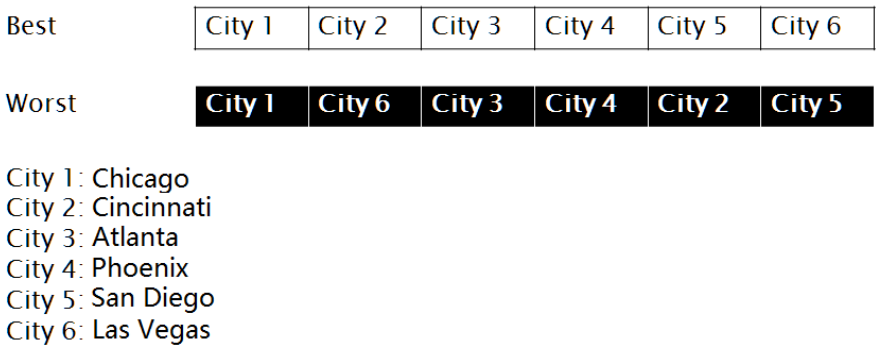

Figure 11: Population containing both best and worst scenario

The simple crossover method which is used in the previous example is not a good fit for TSPs. As we know, a TSP is a typical combinatorial problem. Each feature by itself does not contain any information, but it is rather the sequence of features in an individual that determines its performance. In this case, the simple crossover method mentioned earlier in this chapter will cause two problems. First, it may result in an invalid tour. For example, we should not go to the same city twice in the same trip. Also, we have to travel to all the cities during a single trip without missing any of them, otherwise the tour is invalid. Second, simple crossover is designed for exchanging the features in an individual but not the sequence information contained in an individual. Our goal is to obtain the sequence information from good individuals, then share it with other individuals. Sharing individual features will not help. Because all the individuals contain the same features, the only difference is the sequence of the features.

Sequence information based crossover can solve these issues. In 1991, order-based crossover (OX2) was introduced [45]. This crossover method is designed for scheduling problems and is also suitable for TSPs. Since OX2 is designed for scheduling

problems, the information exchange is based on the sequence information, so there are no invalid individuals generated during crossover.

The procedure of OX2 crossover is as follows.

1. Randomly select several positions in Individual 2. Record the cities in these positions and the sequence of the selected cities.
2. In Individual 1, find the cities recorded in step 1, and record their positions. Replace the cities in these positions in Individual 1 with the same group of cities but in the sequence recorded in step 1.

Here is an example to illustrate how OX2 works. Assume we have two individuals, Individual 1 and Individual 2, which are shown in Figure 12.

Individual 1	City 1	City 2	City 3	City 4	City 5	City 6
Individual 2	City 1	City 6	City 3	City 4	City 2	City 5

Figure 12: Population containing both best and worst Scenario

An example of OX2 crossover is given as follows.

1. Randomly select positions 2, 4, and 6 in Individual 2.
2. The cities in those positions are city 6, 4, and 5.
3. Find the locations of city 6, 4, and 5 in Individual 1.
4. Replace the cities in these locations in Individual 1 in the order 6, 4, 5.

After OX2, the new individual is shown in Figure 13. OX2 crossover created a valid child. This child inherits sequence information from both parents.

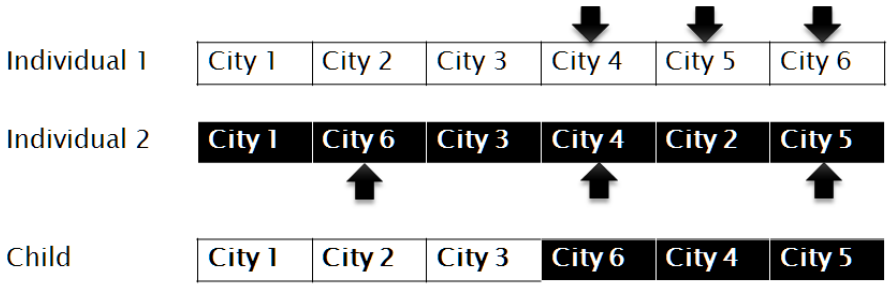

Figure 13: OX2 crossover of two individuals

The creation of a good crossover strategy is based on a good understanding of the problem. The flexibility of heuristic algorithms is its advantage. Standard crossover may still be used in a problem if it does not result in invalid candidate solutions. But we also need to deal with efficiency issues. Consider the diversity of modern science, where each field requires years of learning and training. We cannot provide a simple algorithm to solve all problems in all research areas. A heuristic algorithm can be considered as a tool, or a framework. We need to define details to guarantee correct functioning and satisfactory efficiency. For example, the PID controller is widely used in industry. Assume there is a system containing thousands of PID controllers. How should we tune all the parameters when a system has a complicated structure with a huge number of components? Heuristic algorithms can be an effective approach.

Efficiency is not a strong point for heuristic algorithms, since heuristic algorithms mimic nature and nature is notoriously inefficient. If there is a traditional solution method that can obtain results by solving equations, it should be much faster than most heuristic

algorithms. But if there is a problem with a complex structure and many intractable components, then heuristic algorithms are more efficient. But a single heuristic algorithm is not a panacea for all problems. A good design based on problem specific background knowledge can dramatically improve the efficiency of a heuristic algorithm.

2.2 Analysis of Convergence

Unlike traditional optimization methods, BBO uses evolution to generate new individuals for each generation, which eventually leads to the optimal solution. The ultimate goal for any optimization algorithm is to achieve the global optimum. In contrast to traditional methods, most heuristic algorithms, such as GA and BBO, are considered global optimization methods. The following example illustrates the difference between a global optimization method, and a traditional optimization method which can easily get stuck in local optima – for example, gradient descent with small step size [46]. In the following equation, x is the input (independent variable), and y is the output (cost value). We are looking for the global minimum of y.

$$y = \begin{cases} (x-2.5)^2 + 20, & 0 \le x < 4 \\ (x-6)^2 + 18.25, & 4 \le x \le 10 \end{cases} \tag{2.3}$$

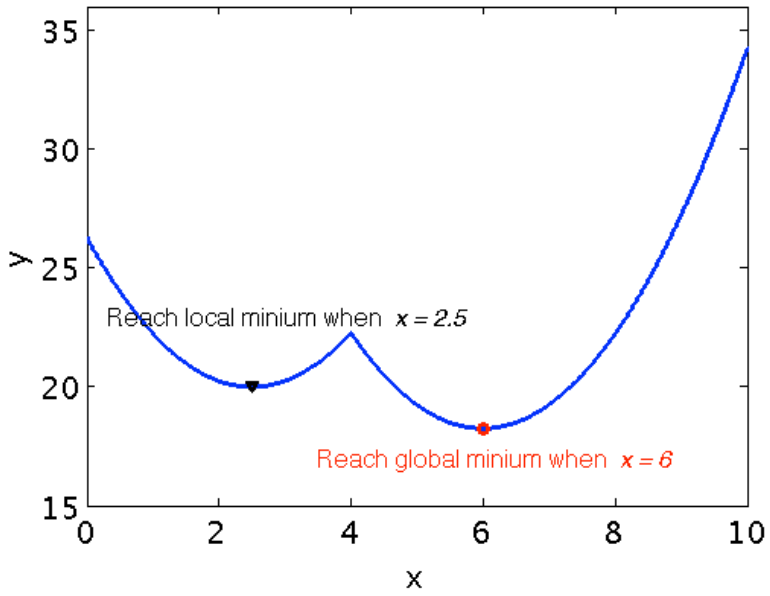

Figure 14: Plot of Equation (2.3) with local and global minimums

Figure 14 shows that the cost contains two minimum values – one is a local minimum and the other is the global minimum. When we apply gradient descent to this problem and search for the minimum value, we may encounter one of the scenarios depicted in Figure 15 or Figure 16.

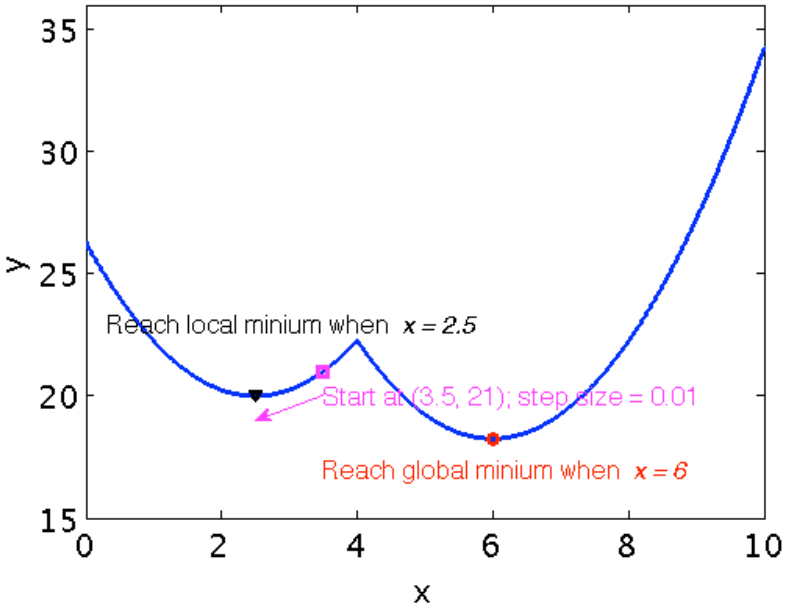

Figure 15: First scenario of gradient descent with different starting points

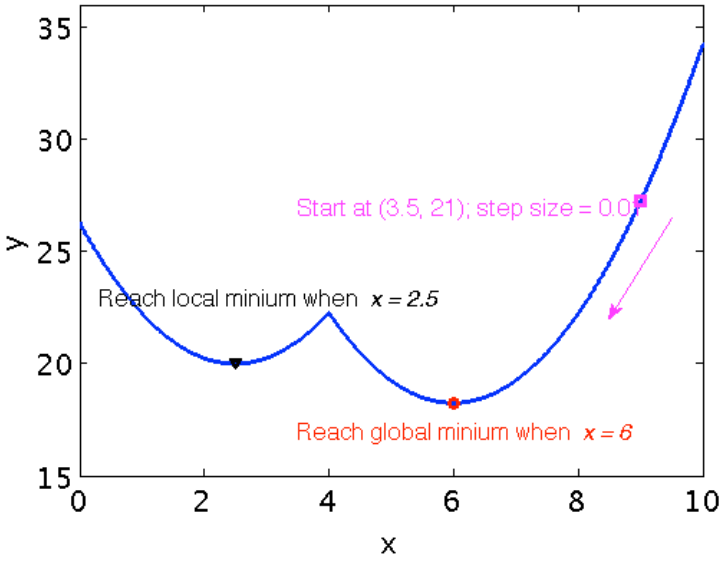

Figure 16: Second scenario of gradient descent with different starting points

Figure 15 and Figure 16 illustrate two common scenarios. If the initial guess point is close to a local minimum and we use a small step size, the algorithm will find a local

minimum instead of the global minimum. In the other scenario the step size is small, but the initial guess is close to the global minimum, so the algorithm can reach it without getting stuck in the local minimum.

Although gradient descent may reach the global minimum, it is still a local optimization method. The same situation applies to most other numerical optimization methods. In contrast to these traditional methods, heuristic algorithms are considered as global optimizers, which are a major advantage compared with traditional optimization algorithms. But there is still a question needed to be answered – will BBO always converge to the global optimal solution?

Markov models are traditional but effective ways to prove the convergence of an algorithm. In [7], a Markov model is derived for BBO. In the following part of this section, we will use the Markov model to perform a convergence analysis of BBO.

In BBO, there are two operations available to create new islands: migration and mutation. We can model these operations to derive the probability that island u becomes to island v after one generation. We make the assumption that each feature type in an island has its own search domain, so migration between islands only happens between the same types of features.

The probability that the s-th feature in island u becomes the s-th feature in island v due solely to migration is

$$\Pr(u(s)=v(s))=\left(1-p_i\right)1_0\left(u(s)-v(s)\right)+p_i\frac{\sum_{j\in J(s)}\mu_j}{\sum_{j=1}^{n}\mu_j} \qquad (2.4)$$

where p_i is migration probability. *J(s)* is the set of islands which contain the same feature as the *s*-th feature in island *v*. μ_j is the emigration rate of island *j*. *n* is the search space size. We can then calculate the probability that individual *v* becomes individual *u* due solely to migration.

$$\begin{aligned}\Pr(u=v) &= \Pr(u_{si})\prod_{s=1}^{k}\Pr(u(s)=v(s)) \\ &= \Pr(u_{si})\prod_{s=1}^{k}\left(\left(1-p_i\right)1_0\left(u(s)-v(s)\right)+p_i\frac{\sum_{j\in J(s)}\mu_j}{\sum_{j=1}^{n}\mu_j}\right)\end{aligned} \quad (2.5)$$

where $\Pr(u_{si})$ is the probability that *u* is selected for immigration, and *k* is the total number of features in each island (that is, the problem dimension).

According to Equation (2.5), we cannot guarantee that island *u* can become island *v* because there is a probability that *J(s)* is null, and at the same time, the *s*-th feature in *u* is not equal to the *s*-th feature in *v*. In that case we obtain Pr*(u=v)* = 0. Since migration can exchange only features that are present in the population between islands, we will not obtain the optimal solution unless all the optimal features are already contained in the feature pool of the population. But this is highly unlikely, especially when a problem is continuous with an infinite number of possible features.

Mutation is another operation that can create new islands. Unlike migration, mutation can create new features which do not exist in the feature pool of the population. The probability that the *s*-th feature in island *u* becomes the *s*-th feature in island *v* due solely to mutation is

$$\Pr(u(s)=v(s))=\left(1-p_m\right)1_0\left(u(s)-v(s)\right)+p_m\frac{1}{n_f} \tag{2.6}$$

where p_m is the mutation rate and n_f is the total number of candidate features in the s-th position. For continuous problems, n_f is infinite in theory. But in practice, there is always a certain problem dependent precision for any numerical method, which means that the total number of possible features will not be infinite, even for a continuous problem. So n_f might be a large number, but it will not be infinite. Then the probability that individual u becomes individual v after one generation can be calculated as

$$\Pr(u=v)=\prod_{s=1}^{k}\Pr(u(s)=v(s)) \tag{2.7}$$

Unlike migration, we obtain $0<\Pr(u=v)<1$. As long as we include mutation in our algorithm, it is guaranteed that $0<\Pr(u=v)<1$ whether we use migration or not. If we use elitism in our algorithm, which means always retain the best island from one generation to another. Then we run BBO for t generations, the probability that we will obtain the optimal solution v is

$$\begin{aligned}\Pr(u=v)&=1-\prod_{g=1}^{t}\left(1-\Pr_g(u=v)\right)\\ \lim_{t\to\infty}\Pr(u=v)&=\lim_{t\to\infty}\left(1-\prod_{g=1}^{t}\left(1-\Pr_g(u=v)\right)\right)=1\end{aligned} \tag{2.8}$$

where, $\Pr_g(u=v)$ is the probability that individual u becomes individual v in the g-th generation. From Equation (2.8), when we run BBO for a large number of generations,

the probability of obtaining optimal island v is close to 1. Since we use elitism in this algorithm, the optimal solution will be retained in the population after we obtain it. According to this convergence analysis of BBO, as long as we include mutation and elitism in BBO, we guarantee convergence to the optimal solution. This shows that BBO, like other EAs, is a global optimization algorithm which provides a significant improvement compared to traditional numerical methods. This proof is true for all heuristic algorithms which contain mutation and elitism.

CHAPTER III

BBO FOR COMBINATORIAL PROBLEMS

3.1 Combinatorial Problems

Combinatorial problems have a finite set of candidate solutions. Usually, exhaustive search is not suitable because of the large size of the problem. The TSP is a classic example of a combinatorial problem. The definition of a TSP is that a salesman has to travel to *c* different cities, so he needs to plan a *c*-city trip and find the shortest, or most time efficient route. In this case, each candidate solution is a combination of cities in some specific order. This is a typical example of a combinatorial problem.

For combinatorial problems, the only guaranteed way to find the optimal solution is by searching through all possible combinations, which is called exhaustive search. But in most cases, the size of the search space is too large for exhaustive search. For example, in a 100-city TSP problem, the number of possible solutions is $100! = 9.33 \times 10^{157}$. That is the reason that we turn to heuristic algorithms as the solution method.

Nature obeys the rule of survival of the fittest. Weak and unhealthy creatures are usually abandoned by nature. Strong and smart creatures can usually obtain more resources and have a better chance of survival. Nature is always dominated by the fittest creatures. Evolution is a process to eliminate weaker species and promote stronger species. Heuristic algorithms like GA, ACO, BBO, and others, are all nature-based algorithms. They all obey the same rule – survival of the fittest. Two things need to be determined in a nature-based algorithm. First, how do we measure the fitness of each individual? Second, how do we improve those individuals?

Fitness is usually easy to determine; for most algorithms, we use an objective function to measure the fitness or performance of an individual. Note that cost and fitness are opposite ways of measuring the same thing. As performance improves, cost decreases, and fitness increases. The second challenge is how to improve the individuals in our algorithm. When translating this into a heuristic algorithm, it means obtaining an efficient population modification method.

The two most common methods to modify a population are recombination (or crossover) and mutation. Crossover is an evolutionary method that involves more than one individual by mixing features from different individuals – it is called migration in BBO. Mutation is a way to modify individuals by introducing randomly selected features. In this chapter, TSP is considered as a representative combinatorial problem. TSPs will be used as benchmark problems to test crossover and mutation methods specially created for combinatorial problems.

3.2 Migration in the Traveling Salesman Problem

BBO migration is the method for combining or modifying features based on parent individuals to create offspring, or new individuals. It is also the most important component in BBO. Combinatorial problems are coded differently than other types of problems. Each element in the individual contains no information by itself, but when we put the elements together in one individual, the order of elements determines the goodness of an individual. To use TSP as an example, an element in an individual is a city. Just knowing that a city is in a tour will not help us determine the distance (or cost) of the entire trip. In order to determine the distance, we need to know the order of all the cities in the tour. Since the original BBO algorithm is not designed for combinatorial problems, we need to modify the original migration methods. Three types of migration methods are introduced: matrix crossover, cycle crossover, and inver-over crossover. These methods have been used in other EAs in past research, but are integrated with BBO here for the first time.

3.2.1 Matrix Crossover

Matrix crossover is introduced by Fox and McMahon in [47]. The advantage of matrix crossover is that it is straightforward and easy to operate. With matrix crossover, an offspring can inherit partial information from both parents, but it will also contain unique information belonging only to itself. The drawback of matrix crossover is that all sequence information is represented by matrices, which requires for a high computational

effort when transforming between tour information in an array expression and tour information in a matrix expression.

The detailed procedure of matrix crossover is as follows. First, for a c-city problem, we need to convert the sequence information of each individual to a $c \times c$ matrix. Each row in the matrix expression provides us the position information of a city in the trip. For example, the k-th row represents the position information of city k. In this expression, each column in a particular row represents a certain city. The number in each column represents the ordering relationship between the column city and row city. For example, if city g is before city k, the number in the g-th row in the k-th column is 1. If city g is after city k, the number in the g-th row in the k-th column is 0. Based on this method, we convert all the individuals in the population to a matrix expression.

Second, based on roulette wheel selection, we select individuals to perform migration. Once the parents are selected, we perform AND logic operation on two parent matrices to obtain the child matrix.

Third, since the child matrix will be incomplete after the previous two steps, we randomly fill in necessary information to create a valid child.

In the last step, we transform the child from its matrix expression to a sequential representation. Figure 17 gives an example of how to apply matrix crossover.

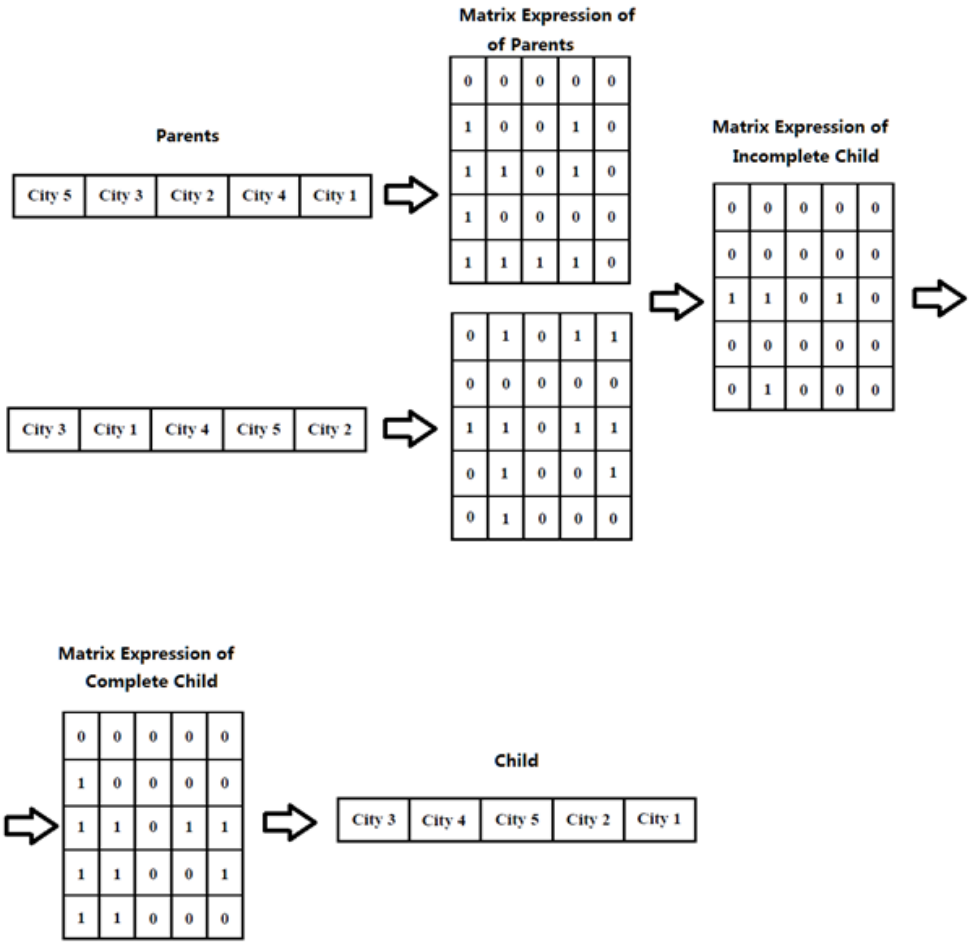

Figure 17: Example of matrix crossover with a 5-city TSP

3.2.2 Cycle Crossover

Cycle crossover has been tested in [48], and resulted in superior performance against competitors. It also achieved satisfying results in [26]. In this section, it is used as the default migration method in BBO. The application of cycle crossover is fairly easy. In contrast to matrix crossover, no tour format transformation is needed, and cycle crossover guarantees that every child is valid and complete. For these reasons, cycle crossover has been widely used in EAs for combinatorial problems.

The basic procedure of cycle crossover is as follows.

1. We randomly select a city as the starting point in parent 1 and record its position.
2. In parent 2, we find the city in the position recorded in parent 1, and then record this city. We go back to parent 1, search for the city we found in parent 2, and then record its position in parent 1.
3. We repeat step 2 until we obtain a closed cycle, which means that we have returned to the starting city. Then we copy the cities from the closed cycle in parent 2, and the cities that are not in the closed cycle in parent 1, to obtain child 1. Similarly, we copy the cities from the closed cycle in parent 1, and the cities that are not in the closed cycle in parent 2, to obtain child 2.

We provide an example in Figure 18 to illustrate the application of cycle crossover. In this figure, city 2 in parent 1 is randomly selected as the starting point city. Following the basic procedure of cycle crossover, the closed cycle we find is city 2 - city 1 - city 4 - city 8 - city 3 - city 2.

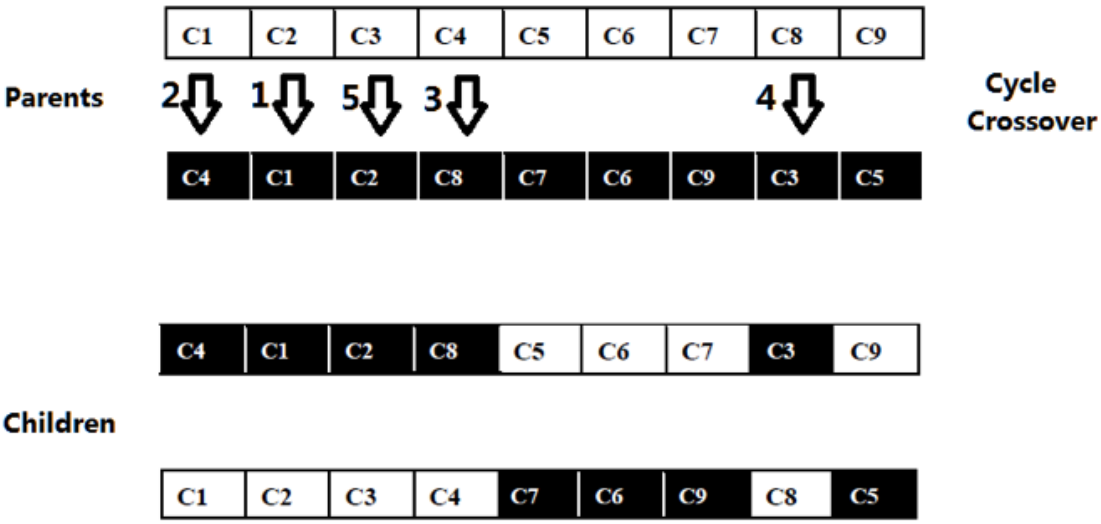

Figure 18: Example of cycle crossover with 9-city TSP

3.2.3 Inver-over Crossover

The third migration method is called inver-over crossover, originally invented in [49]. Like cycle crossover, all the children generated by inver-over crossover are guaranteed to be valid and complete. Inver-over crossover does not require any expression transformation. The basic procedure of inver-over crossover is as follows.

1. Randomly select a city in parent 1 as the starting point, which is called city *s*.
2. Find city *s* in parent 2 and choose the city that follows it as the ending point, city *e*. Then find city *e* in parent 1.
3. Reverse the cities between the city following city *s*, and city *e*, in parent 1. An example is provided in Figure 19.

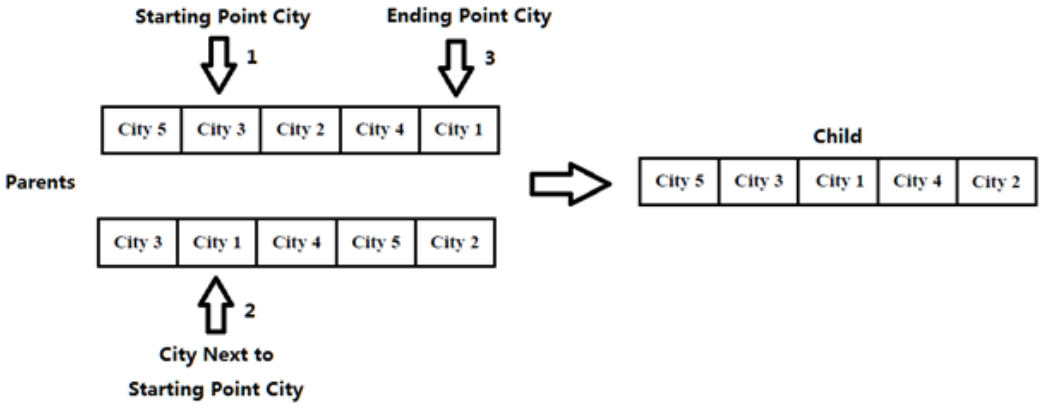

Figure 19: Example of inver-over crossover with 5-city TSP

3.3 Local Search Optimization

Combinatorial problems make good benchmarks because of their special characteristics. For example, most benchmarks are composed of variables, and each variable has its own search domain. In that case, heuristic algorithms need to search each variable in its own domain for the optimal solution. But combinatorial problems are different. Again, we use TSP as an example, in which the coordinates of each city are fixed. The task of heuristic algorithms is to rearrange the order of the cities and search for the optimal solution. In other words, each individual in the population has enough information to create an optimal solution.

Local search optimization can find the optimal solutions only by modifying an individual candidate solution. For TSP, all the necessary cities to create an optimal solution are contained in each individual. Since the combination of techniques can be more powerful than a single technique [13], in the BBO modification in this section, we

will introduce local search as a complement to migration. In the next part of this section, we introduce three local optimization methods which have been successfully implemented in TSPs: 2-opt, 3-opt, and *k*-opt. These methods are applied after migration as a complement to our migration strategy.

3.3.1 2-opt and 3-opt

2-opt is a simple but effective local research method invented in 1958 [50]. Although this method is easy to operate, it shows good performance with TSPs. The operation of 2-opt is as follows.

1. Find a random individual in the population.
2. Break two links in this individual.
3. Randomly connect the cities which only have one link connected, with the constraint that the resulting path includes all cities.

In Figure 20, we apply 2-opt to an 8-city TSP.

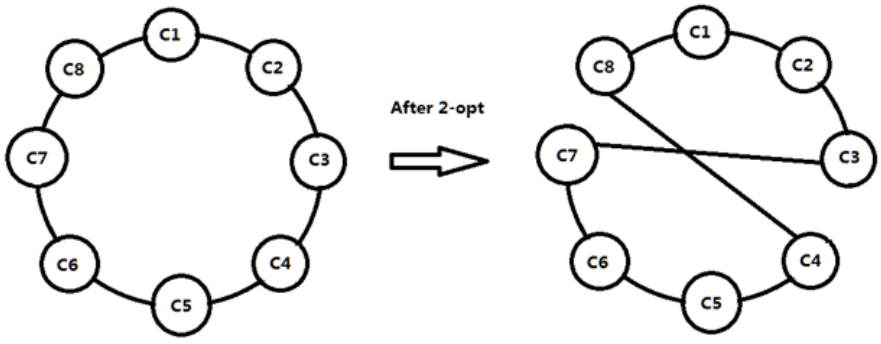

Figure 20: Example of 2-opt with 8-city TSP

3-opt is an updated technique based on 2-opt [50]. Instead of replacing two links in the individual, 3-opt will break three links then randomly reconnect the cities. Even though 2-opt and 3-opt have good performance in sequence-based problems, their disadvantage is obvious: the number of the links to break and reconnect is predefined, and does not adapt to the problem.

3.3.2 *k*-opt

In order to address the disadvantage of 2-opt and 3-opt, *k*-opt was introduced and discussed in [50]. *k*-opt is a method for adaptively choosing the number of links to break and reconnect. According to experimental results, when the number of replaced links increases, the performance of *k*-opt increases. But the computational burden also increases. We need to find a balance between the expected performance and the computational burden. For the first few optimization generations, the population is still diverse and there is a lot of information for migration to exploit. In this case, we do not need *k*-opt to act aggressively, so *k* should be a small number. But as the optimization algorithm progresses, the algorithm begins to converge. In this situation, we need to increase the effectiveness of *k*-opt to increase the rate of population improvement, so we should use a bigger *k* value. We conclude that *k* should increase as the generation count increases. One way of doing this is shown as follows.

$$k = \left\lfloor \frac{g_c c}{2 g_m} \right\rfloor \tag{3.1}$$

c: number of cities.

g_m: maximum generation number.

g_c: current generation number

3.4 Population Initialization and Greedy Method

Migration, mutation, and local search optimization are the three main components in most BBO implementations. Their roles are to improve the performance of the entire population. We can also increase the rate of population improvement by applying new techniques in the heuristic algorithm. In this section, we will focus on using a modified population initialization algorithm, and using greedy methods, to increase the rate of population improvement.

3.4.1 Population Initialization

Population initialization is usually the first step for heuristic algorithms. Most of the systems we apply heuristic algorithms to have complex structures. We do not have a good understanding of the effect of each independent variable in those systems. That means we do not know how to create an initial population based on our expertise, so instead we randomly create it. This is no doubt the simplest method for population initialization. However, it is also the most inefficient way to create an initial population. Fortunately, random population initialization is not the only method for population initialization; for certain problems like the TSP, there are certain ways of creating an initial population which can provide a great benefit to EA performance.

For TSPs, the most commonly used technique is NNA [44]. The detailed procedure is as follows.

1. Randomly select a city as the end point of the trip (it is also the starting point).
2. Calculate the distance between the end point city and the cities which are not included in the trip. The first time through this loop, this will include all cities except for the starting/end point city.
3. Based on the distances calculated in step 2, find the nearest city to the end point city. Link these two cities, and name the most recently added city as the end point city.
4. If all the cities are included in the trip, terminate; otherwise, go to step 2.

The procedure is fairly easy to operate, and is clearly not time consuming even for large-scale problems. The most time consuming part is the calculation of the Euclidean distances between cities. For a TSP with c cities, the total number of calculations of Euclidean distance is

$$\text{number of calculations} = \frac{c(c-1)}{2} \tag{3.2}$$

For a 1000-city problem, the total number of calculations is 499,500, which is an acceptable number when considering the problem size.

3.4.2 Greedy Methods

Greedy methods have a long history as effective techniques in heuristic algorithms, and many algorithms use it as a basic component. The definition of greedy methods is implied by its name: always choose the short-term benefit, and refuse to

accept any short-term losses [51]. Being greedy at every step may not be the best choice for all situations. But in some problems, like the TSP, greedy methods can be a helpful complement to optimization algorithms.

In BBO, we can use greedy methods in three places – migration, local optimization, and mutation. As we know, migration is a function for an individual to share information with other individuals to generate offspring. Although the individuals with better performance have higher probabilities to share features, and the individuals with worse performance have high probabilities to import features, there is no guarantee that the child will outperform its parents. In this way, local optimization methods are similar to migration; we cannot guarantee that offspring perform better than their parents. Mutation introduces random information to the population. New individuals have unpredictable performance in this case. Should we need to keep the offspring with worsened performance? If the diversity of the population is low, even though the offspring has worse performance than either of the parents, we may still want to keep it. But if we want the performance of the entire population to improve in the short term, then we should use greedy methods to abandon offspring with worse performance.

3.5 TSP Simulation

In this section, the techniques mentioned in the previous sections are tested on four TSP benchmarks. All the benchmarks are selected from the standard TSP benchmark library TSPLib [43]: ulysses16, st70, rat575, and u2152. In order to obtain a broad

comparison between techniques, benchmark sizes vary from small to extra large problems. ulysses16 is a 16-city TSP; st70 is a 70-city TSP; rat575 is a 575-city TSP; and u2152 is a 2152-city TSP.

In this dissertation, the aim is not only to find the best modification of BBO, but also to compare BBO with other popular optimization algorithms. Four popular competitors are selected: GA [52], NNA [44], ACO [53], and simulated annealing (SA) [54]. In order to guarantee fairness in our comparisons, we set two common termination criteria for each algorithm. The algorithm will terminate when either of them is met.

- Number of evaluations of cost function: 10,000
- CPU time: 300 sec

Also, since the performance of heuristic algorithms varies from simulation to simulation due to their stochastic nature, a single simulation may not reflect the true performance of an algorithm. To guarantee a fair comparison, Monte Carlo simulations are performed. We conduct each simulation d times, and take the average performance as the overall performance metric. Here, we use $d = 20$.

In order to compare the performance of different techniques, we use the default BBO setup for each of the BBO modifications for all the components that are unmodified. The default BBO setup is as follows.

- Population size: 100
- Number of elite individuals per generation: 1
- Population initialization: Random
- Migration: Cycle crossover

- Mutation rate: 0.01
- Local optimization method: None
- Greedy Methods: None

3.5.1 Population Initialization

First we test the performance of different population initialization methods. We designed six population initialization methods: no NNA; NNA for 1 individual; NNA for 5 individuals; NNA for 50 individuals; NNA for 75 individuals; and NNA for 100 individuals (the entire population). The simulation results are shown in Table i.

	Best distance and CPU time per simulation (sec)						
		No NNA	1 NNA	5 NNA	50 NNA	75 NNA	100 NNA
ulysses16	Distance SD	75.68 0.40	74.72 0.68	**74.23** 0.55	74.65 0.32	74.66 0.56	74.62 0.25
	CPU Time	**3.11**	3.13	3.13	3.21	3.24	3.25
st70	Distance SD	1432 29.07	728 22.51	729 32.99	727 33.57	726 27.27	**725** 31.20
	CPU Time	**4.23**	4.55	4.56	4.61	4.62	4.65
rat575	Distance SD	128090 2608.86	54487 1235.90	54483 1370.21	**54481** 1312.16	54485 1249.56	54482 1458.13
	CPU Time	**19.23**	19.31	19.34	19.37	19.56	19.58
u2152	Distance SD	241745 2162.19	74355 790.00	**74322** 838.27	74323 809.13	74333 952.63	74325 746.03
	CPU Time	**40.23**	41.92	42.35	42.58	42.62	42.66

Table i: Performance of NNA in BBO, the best results averaged over 20 Monte Carlo simulations, and the standard deviations of the best distances. The best results in each row are shown in bold font.

When compared on the basis of computation time, No NNA is the quickest. But the performance difference between no NNA and 1 NNA is large, especially for larger scale problems. When we apply NNA to the BBO algorithm, the performance between different setups is very similar. The standard deviations show that BBO performs significantly better with NNA than without NNA. Based on the simulation results, the best overall setup is 1 NNA, which means NNA is only used on one individual.

3.5.2 Crossover Methods

Next we test different crossover methods. Three crossover methods were discussed earlier: matrix crossover, cycle crossover and inver-over crossover. Their performances are shown in Table ii.

	Best distance and CPU time per simulation (sec)			
		Matrix	Cycle	Inver-Over
ulysses16	Distance SD	74.22 0.55	75.68 0.40	**74.21** 0.32
	CPU Time	**0.64**	3.11	0.97
st70	Distance SD	2725 86.09	1432 29.07	**820** 17.34
	CPU Time	2.22	4.23	**1.05**
rat575	Distance SD	102763 3006.57	128090 2608.86	**78765** 1657.42
	CPU Time	300.00	19.23	**2.93**
u2152	Distance SD	434209 5620.58	241745 2162.19	**237372** 2012.46
	CPU Time	300.00	40.23	**10.23**

Table ii: Performance of matrix crossover, cycle crossover and inver-over crossover, the best results averaged over 20 Monte Carlo simulations, and the standard deviations of the best distances. The best results in each row are shown in bold font.

The simulation results show that inver-over crossover dominates the other methods on all the benchmarks, both in terms of performance and computation time. It also has smaller standard deviations for all benchmark problems. Also, the computation time of matrix crossover becomes very long when the problem size increases, so it is not a good choice for large-scale problems.

3.5.3 Local Optimization

Next we evaluate local optimization methods. Three methods were proposed: 2-opt, 3-opt, and k-opt. When using local optimization, we apply the local optimization to each individual in the population at the end of each generation. The performances of the different local optimization methods are shown in Table iii.

The setup with the best computation time is BBO without local optimization methods. Despite the small increases in simulation time, improvement in performance is obvious when using local optimization. For a small size problem, 2-opt and 3-opt outperform *k*-opt, but with large-scale problems, *k*-opt is the best choice.

TSP	Best distance and CPU time per simulation (sec)				
		No-opt	2-opt	3-opt	*k*-opt
ulysses16	Distance SD	75.68 0.40	74.67 0.37	**74.65** 0.47	80.59 0.41
	CPU Time	**3.11**	3.18	3.23	3.57
st70	Distance SD	1432 29.07	**1180** 28.23	1695 31.87	1773 29.76
	CPU Time	**4.23**	5.67	6.72	7.55
rat575	Distance SD	128090 2608.86	100069 2235.98	97759 1781.32	**94763** 1341.92
	CPU Time	**19.23**	25.45	27.56	30.01
u2152	Distance SD	241745 2162.19	240001 1943.12	235987 1903.74	**235876** 1788.56
	CPU Time	**40.23**	54.34	58.31	153.99

Table iii: Performance of No-opt, 2-opt, 3-opt and k-opt, the best results averaged over 20 Monte Carlo simulations, and the standard deviations of the best distances. The best results in each row are shown in bold font.

3.5.4 Greedy Methods

Now we test different greedy method setups. Three setups are introduced: first, no greedy method; second, half of the population uses a greedy method (the individuals that use greedy methods in this approach are randomly selected); third, the entire population uses a greedy method. In all three of the setups, we apply the greedy method to migration, local optimization, and mutation. The performances of different greedy method setups are shown in Table iv.

TSP	Best distance and CPU time per simulation (sec)			
		No Greedy	Half Greedy	All Greedy
ulysses16	Distance SD	**75.68** 0.40	79.41 0.35	88.51 0.32
	CPU Time	**3.11**	3.12	3.15
st70	Distance SD	**1432** 29.07	1770 34.34	2795 52.86
	CPU Time	**4.23**	4.62	4.73
rat575	Distance SD	128090 2608.86	**10360** 1863.21	10456 1897.96
	CPU Time	**19.23**	19.35	19.47
u2152	Distance SD	241745 2162.19	242356 2129.75	**23632** 1736.12
	CPU Time	**40.23**	42.44	43.12

Table iv: Performance of different greedy method setups, the best results averaged over 20 Monte Carlo simulations, and the standard deviations of the best distances. The best results in each row are shown in bold font.

These simulation results tell us that greedy methods slow down the optimization process in general. For small TSP sizes, greedy methods reduce performance; however, for larger TSP sizes, greedy methods improve performance.

3.5.5 Comparison with Other Algorithms

Based on the previous simulation results, the best overall setup for BBO is the following: 1 NNA for population initialization; inver-over crossover; *k*-opt for local optimization; and all greedy for the greedy method setup. Here, we compare the results between BBO, GA, NNA, ACO, SA, and Modified BBO for TSPs (BBO/TSP). The setups of these algorithms are as follows.

- GA: Population size is 100; Crossover is a combination of flip crossover, swap crossover and slide crossover; Crossover rate is 0.5; Mutation rate is 0.01.
- NNA: It is not a heuristic algorithm, so no tuning parameters are needed.
- ACO: Population size is 20 ants; Initial pheromone value is 10^{-6}; Pheromone update constant is 20; Exploration constant is 1; Global pheromone decay rate is 0.9; Local pheromone is decay rate 0.1; Pheromone sensitivity is 1; Visibility sensitivity is 1.
- SA: Initial temperature is 2000; Maximum trails at a temperature is 10 times the population size.
- BBO/TSP: Population size is 100; Number of elite individuals per generation is 1; Population initialization is 1 NNA; Migration method is inver-over crossover; Local optimization method is *k*-opt; Greedy method is all greedy.

TSP	Best distance and CPU time per simulation (sec)						
		GA	NNA	ACO	SA	Default BBO	BBO/TSP
ulysses16	Distance SD	74.63 0.49	104.43 0.16	74.62 0.61	74.77 0.45	75.68 0.40	**74.21** 0.34
	CPU Time	3.41	**0.18**	0.38	1.01	3.11	5.12
st70	Distance SD	1509 37.95	3208 120.50	1359 48.12	**741** 24.16	1432 29.07	802 23.64
	CPU Time	6.22	**0.19**	4.47	3.98	4.23	5.21
rat575	Distance SD	12493 244.32	12952 352.39	68311 1861.99	**12399** 255.10	128090 2608.86	76321 1131.00
	CPU Time	11.12	**0.24**	300.00	8.18	19.23	24.32
u2152	Distance SD	82205 1301.57	82209 1590.31	150341 3117.80	709209 8188.67	241745 2162.19	**77828** 786.49
	CPU Time	18.45	**0.67**	300.00	23.16	40.23	6.04

Table v: Performance of GA, NNA, ACO, SA, default BBO and BBO/TSP, the best results averaged over 20 Monte Carlo simulations, and the standard deviations of the best distances. The best results in each row are shown in bold font.

Based on the simulation results in Table v, in ulysses16, BBO/TSP achieved the best overall solutions. Although the computation time is slightly longer than the others, it is still tolerable. In st70, SA has the best performance, and BBO/TSP has the second best, which is close to the results from SA, and far better than others. In rat575, SA is the best choice as far as the solution quality, but it is more time consuming compared to NNA, and BBO/TSP only has fair performance on this benchmark. With the largest benchmark, u2152, BBO/TSP achieved the best performance and fastest convergence speed among all of the heuristic algorithms. According to these results, BBO/TSP has the best overall performance.

3.6 BBO GUI for TSP

According to the results from the previous section, hybrid BBO can be much more effective than its predecessors. Hybridization has become the trend for algorithm design. The key to designing a hybrid algorithm is that each component in the algorithm should be fairly independent from the others. In other words, the algorithm is a well-designed framework with a modular design pattern. So each component is an independent module. Components like population initialization, crossover, greedy methods, and local optimization, are considered as modules in the algorithm. With a standard input/output (I/O) interface, each module can be easily replaced with alternative, newly designed modules.

Algorithm modulation can benefit researchers when attempting new techniques. It requires minimal modification to implement different algorithmic techniques with a well-designed I/O interface. Because of the popularity of BBO, numerous hybrid algorithms have been developed [13] [14] [15]. But there is no consistent format for BBO algorithms, so the effort to implement new techniques into BBO can be significant. In this section, we will introduce a GUI for BBO as applied to TSPs. Also, we will introduce a modularized format for BBO.

3.6.1 Module Categories in BBO

Before designing an algorithm, there is a question we need to address – what is the structure of the algorithm? Since we need to design a modularized BBO algorithm,

the answer to this question should be divided into two parts: how to design modules, and how to connect modules.

There are five module categories in BBO for TSP:

1. BBO framework
2. Population initialization
3. Recombination
4. Local optimization
5. Greedy methods

The BBO framework is the most important module category. This category only includes one module, which is called the BBO framework module, and it cannot be replaced by an alternative module. This module contains the fundamental BBO algorithms and defines the interface with the other modules. All the other modules need to be connected to the BBO framework module in order to function correctly.

Since the BBO framework module serves as the interface for the entire algorithm, it needs to provide standard connections to other modules. We prefer a plug-and-play system, so every module is independent from each other, and the communication between modules are solely based on the I/O interfaces in the BBO framework modules. The I/O format for each module besides the BBO framework module is shown as follows.

- Population initialization

Inputs: 1) Coordinates of all cities; 2) Number of SIVs per individual (i.e., number of TSP cities); 3) Randomly generated city order for each individual.

Output: 1) city order for each individual after applying a population initialization technique.

- Recombination

Inputs: 1) coordinates of all cities; 2) tour distance of each individual; 3) number of individuals in the population; 4) number of SIVs per individual (i.e., number of TSP cities); 5) city order for each individual.

Outputs: 1) city order for each individual after applying a recombination method; 2) tour distance of each individual after applying the recombination method.

- Local optimization

Inputs: 1) coordinates of all cities; 2) city order for each individual; 3) Current generation number.

Outputs: 1) city order for each individual after applying the local optimization method; 2) tour distance of each individual after applying the local optimization method.

- Greedy method

Inputs: 1) the city order for each individual; 2) the tour distance of each individual; 3) the city order of each individual; 4) the tour distance of each individual.

Outputs: 1) the city order for each individual after applying the greedy method; 2) the tour distance of each individual after applying the greedy method.

3.6.2 Default Modules

Population initialization is the second module category. The purpose of this module category is to preprocess the population before applying the BBO algorithm. There are many traditional methods which can significantly increase the quality of the entire population without much computational effort. Although we cannot obtain the optimal solution or even be close to the optimal solution, our goal here is only to provide a better start for BBO. Based on the simulation results from the previous section, NNA can provide a high quality initial population for BBO, and lead to better final results. In this category, researchers can also design their own preprocessing algorithms following the I/O format described above. In the BBO GUI for the TSP, we provide five default modules for population initialization: NNA0, NNA1, NNA5, NNA10, NNA100. The numbers in the module names represent how many individuals will be initialized with NNA. For example, NNA1 means that we only perform NNA on one individual.

Recombination (or crossover) is the most critical component in BBO. The same algorithm with different crossover methods can have very different performances on the same benchmark problem. In this case, a crossover upgrade might be the main focus for an algorithmic modification. For most algorithms, crossover methods are deeply embedded in the algorithms. So in order to test different crossover methods, algorithms need to be rewritten most of the time. The goal of our GUI design is to provide a platform so researchers can switch between different techniques with minimal effort. As long as the crossover modules follow the I/O format described above, nothing else needs to be changed after plugging them into the BBO framework module. The default crossover

modules include: matrix crossover module, cycle crossover module, and inver-over crossover module.

Local optimization is a complementary technique for crossover. The focus of local optimization is local search rather than global search. The default modules of local optimization include: opt2 module (perform 2-opt), opt3 module (perform 3-opt), opt*k* module (perform *k*-opt).

Greedy methods are very effective in some cases, so we also include it as a module category. Researchers can develop different greedy strategies to achieve the best results. There are four default greedy modules provided in this GUI: greedy0 module, greedy1 module, greedyhalf module and greedyall module. The greedy0 module does not implement any greedy method. The greedy1 module implements a greedy method on one individual. The greedyhalf module implements a greedy method on half of the population. The greedyall module implements a greedy method on all the individuals.

3.6.3 TSP GUI based on BBO

The BBO GUI is built with the modules mentioned in the previous subsections. Since the entire GUI interface is too large to display on a single figure, we discuss the interface one piece at a time in this section. The first part of the GUI is the TSP benchmark selection, which is shown in Figure 21.

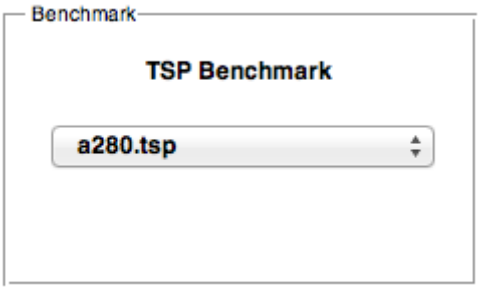

Figure 21: The BBO benchmark selection

In this GUI, there are a total of 100 TSP benchmark problems, and users can choose any of them from the menu shown in Figure 21.

The second part of the GUI is the BBO setup, which contains two user-defined parameters - population size and generation limit. Those two parameters are problem dependent, and users can choose appropriate values based on their experience. The BBO setup window is shown in Figure 22.

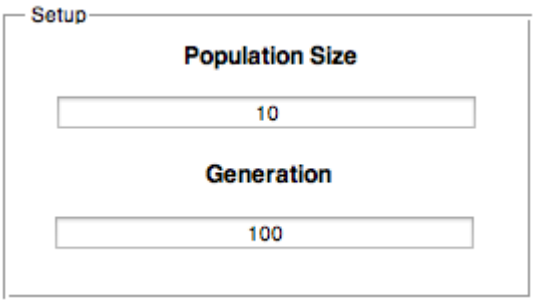

Figure 22: The BBO setup selection

The third part of the GUI is the BBO technique module, which includes population initialization, migration, local optimization, and greedy method. Each module

contains several options. Any user-created module will automatically be displayed as an option in the corresponding module category.

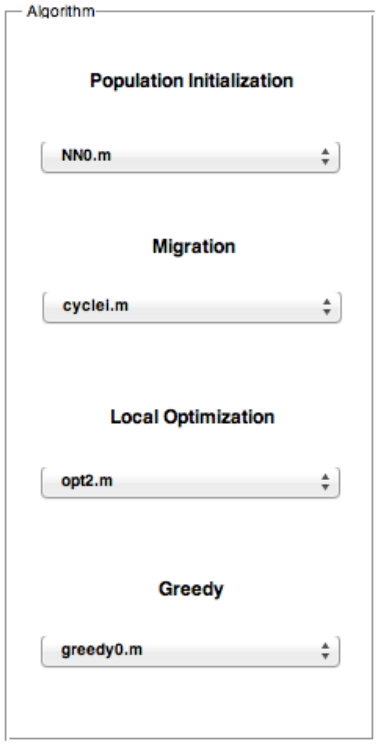

Figure 23: The BBO technique selection

The fourth part of the GUI is the control panel, and it includes the plot selection menu, the run button, and the clear button. This GUI contains four plot locations. The user can choose any of them for their cost vs. generation plot. The reason we decide to provide four plots in the GUI is because we encourage users to draw plots at different locations with different selected techniques. Side by side comparison is the most intuitive way to visualize the performance comparison of different techniques. The run button is used to begin the BBO algorithm. The clear button is used to clear the selected plots from the GUI.

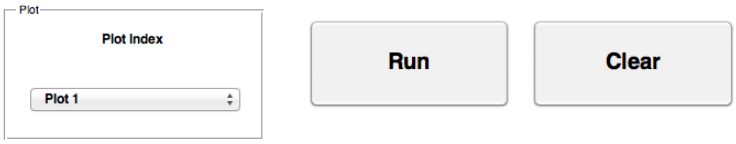

Figure 24: GUI control panel

The fifth part of the GUI is the plot section, which includes four plots. Users can select any of those locations to draw the output plots from a given BBO simulation.

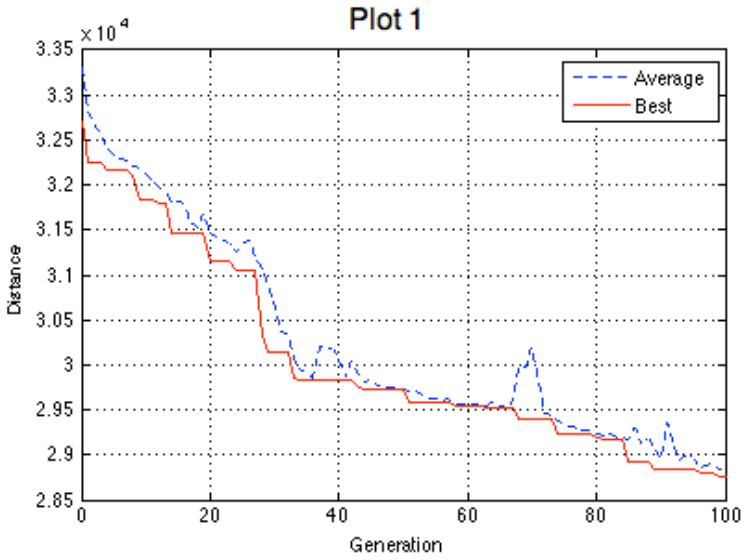

Figure 25: Plots in GUI

The sixth part of the GUI is the function panel. You can save figures, save data from figures, and access the help file from this panel.

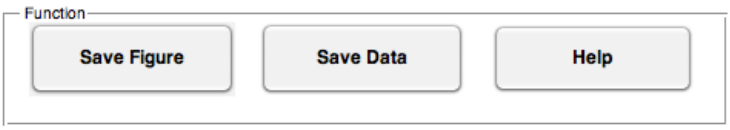

Figure 26: Function panel of GUI

The last part of the GUI is the TSP map. Plotting a TSP map to display the best solution at each generation is an intuitive way to visualize the improvement of the best BBO solution from one generation to the next.

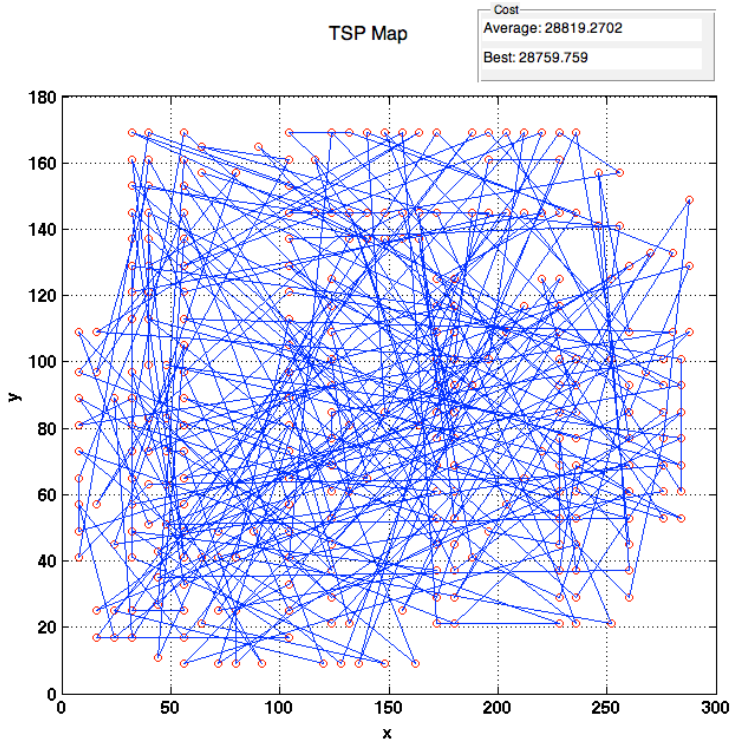

Figure 27: TSP map of GUI

With this GUI, users can easily implement different techniques on the 100 benchmark problems that have been provided, and can also add their own TSPs. Also,

based on the standardized I/O interface, new techniques can be implemented with minimal effort.

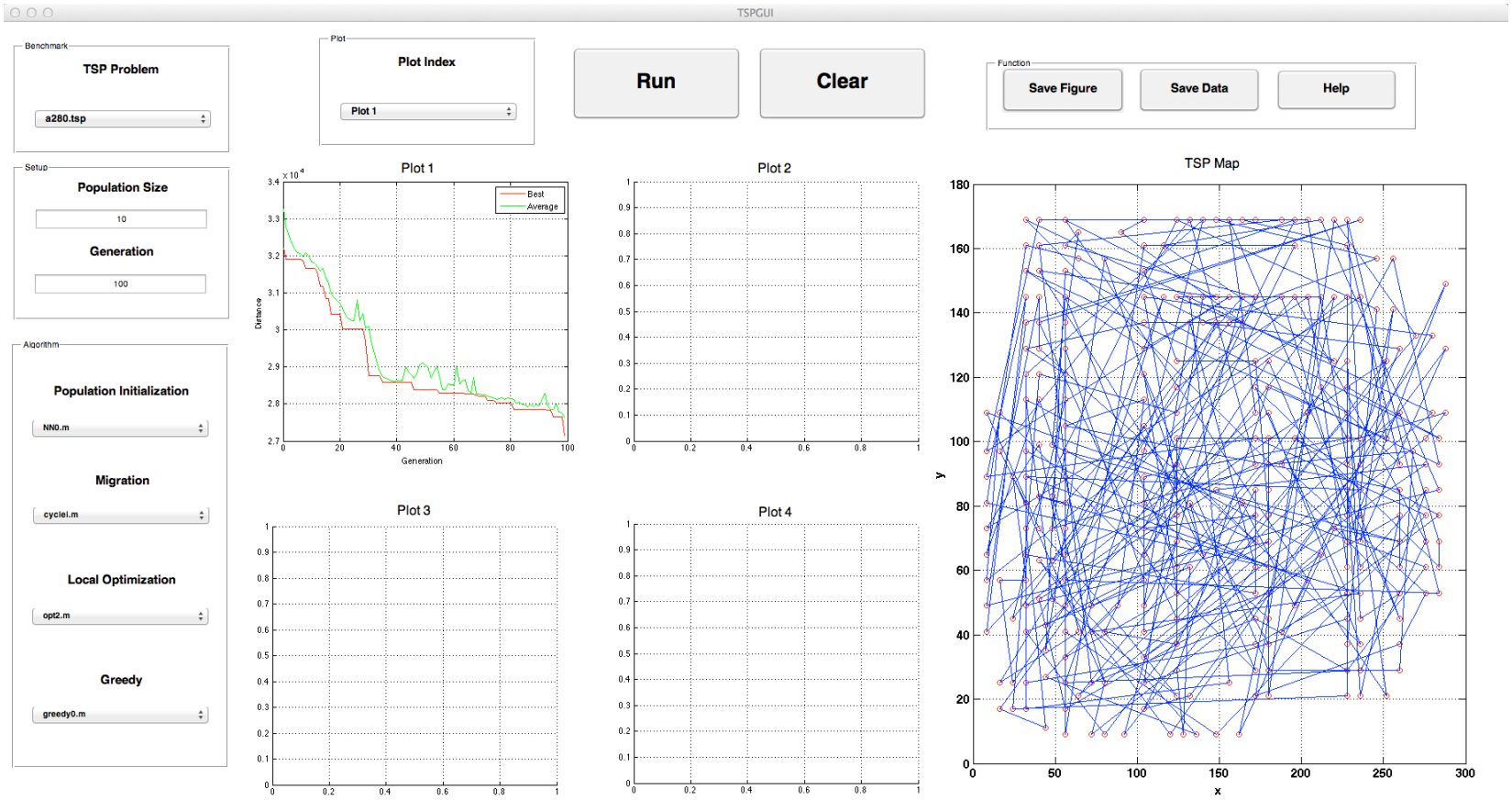

Figure 28: BBO GUI for TSPs

CHAPTER IV

COMPLEX SYSTEM OPTIMIZATION

The material in this chapter is based on [28], which is one of the dissertation author's publications. It is used here with permission. Optimization problems with complex structures are hard to solve. For a nonlinear problem with multi-objectives and multi-constraints, a heuristic algorithm is a good option because of its flexibility and ease of implementation. For real world engineering applications, we find that few systems are simple. Most consist of several interacting subsystems, each of which has multi-objectives and multi-constraints. The optimization of a complex system is a challenge because we cannot treat each subsystem separately. The selected optimization methods need to consider the entire system. Since the subsystems are not totally independent from each other, it is ideal if we can combine their optimization by synchronizing the local and global optimization procedures.

4.1 Structure of Complex Systems

In modern industry, system structures are complex. It is hard to find a system with only one input, one output, one objective, and no constraints. Instead, multi-inputs, multi-outputs, multi-objectives, and multi-constraints are common. Modularity has also become common in industrial design for several reasons. First, the maintenance of modularized systems is relative easy. Problem diagnosis can be localized in each component. Second, updating such systems will not affect the entire system. We also find that adding more components, or replacing components in a modularized system, can be easily accomplished. A complex system is usually a modularized system, with a structure that consists of multi-modules. The only connection between each module is the parameter inputs and outputs. In other words, a complex system consists of relatively independent subsystems. Each subsystem has its own inputs, outputs, objectives, and constraints.

We use automobile assembly processing as an example of a complex system. A manufacturing plant is usually configured to assemble more than one model of vehicle. Each model built in the plant can be considered a subsystem. Each subsystem is different. For this reason, each subsystem can be treated as an independent system. But these subsystems still belong to the same system, and usually they have some aspects in common. For example, they may share similar objectives, like compatibility of parts, total cost of each vehicle, or assembly costs. This same situation also applies to the constraints. For example, the material cost and labor cost are common constraints in the assembly process. In this case, it is not necessary to optimize each subsystem individually. Although the subsystems are not identical to each other, they still share similar objectives

and costs. So information exchange between subsystems is mutually beneficial to every subsystem. Without individually optimizing for each subsystem, we treat all subsystems as one integrated system. We can now study the global optimization of all subsystems, which is referred to as the optimization of a complex system.

4.2 Algorithms for complex system optimization

Some problems, such as TSPs, truck routing, or sensor selection, are combinatorial, or cannot be characterized by equations. Optimization methods such as Newton's method or gradient descent are not suitable in this case. Other than using brute-force search, a heuristic algorithm is the best possibility that remains.

In a complex system, each subsystem has its own objectives and constraints. When compared with simple systems, the complex system has three extensions: from single objective to multiple objectives, from no constraints to multiple constraints, and from optimizing only one system to optimizing multiple subsystems. With a problem involving multi-objectives and multi-constraints, two types of techniques are used to deal separately with each objective and constraint. The final results are then calculated based on the combination of these techniques. When we are confronted with the problem of optimizing multi-systems, we enter new territory relative to traditional optimization theories. It is not only a theoretical achievement to solve these types of problems, but also a significant contribution to the industry. Multi-objectives and multi-constraints represent most of the problems faced by industry today. If we can optimize similar problems all at

once, rather than one at a time, we can significantly increase the efficiency of the optimization process.

Based on its performance on benchmarks [1], we decide to implement BBO for complex systems. Since complex systems are much different than simple systems, due to having multi-systems, multi-objectives, and multi-constraints, we need to change the structure of BBO so that it is suitable for complex systems [55]. The major change is in migration. First, the ranking strategy is different because we need to assign ranks to individuals based on the performance of all objectives. Also, migration becomes more complex, because we need to migrate both within single subsystems, and also between subsystems. In addition to the change in migration, other parts of BBO are also changed. One such change is with regard to the population setup. Since we have more than one subsystem, there is more than one subpopulation contained within the population. The modification of BBO for complex system optimization will be discussed in detail in the following sections of this chapter.

4.3 BBO for Complex Systems

In this section, we focus on the modification of BBO for complex systems. The following features of complex systems must be considered: the multi-subsystems structure, the multi-objectives of each subsystem, and the multi-constraints of each subsystem.

The original BBO algorithm was designed for a single objective, no constraints, and single-system problems. But since then, BBO has been extended to multi-objective problems [56] and multi-constraint problems [57]. As we recall from Chapter 1, the main feature of complex systems is its multi-subsystem structure. Therefore, our major goal is to extend BBO to systems with multi-subsystems, where each subsystem contains multi-objectives and multi-constraints. Our new algorithm is called BBO/Complex.

Our first BBO extension involves its environment, or its population structure. The original BBO environment is an archipelago that consists of islands. The islands represent possible solutions to the problem. This BBO environment is based on the premise that BBO is a single system optimization algorithm. Complex systems contain more than one subsystem, each of which is partially independent from the others. Therefore, the environment of BBO/Complex includes n archipelagos, where n is the number of subsystems. The second difference between BBO and BBO/Complex involves objectives and constraints. The original BBO algorithm only includes one objective and no constraints, but BBO/Complex includes multi-objectives and multi-constraints. The new environment of BBO/Complex is as follows [55].

1. $P = \{A_1, A_2, A_3, ...\}$ is a population that is comprised of archipelagos. Each archipelago corresponds to one subsystem.

2. $A_h = \{I_{h1}, I_{h2}, I_{h3}, ...; O_{h1}, O_{h2}, O_{h3}, ...; C_{h1}, C_{h2}, C_{h3}, ...\}$ is an archipelago that is comprised of islands I_{hi}, objectives O_{hi}, and constraints C_{hi}.

3. $I_{hi} = \{S_{hi1}, S_{hi2}, S_{hi3}, ...\}$ is an island that is comprised of SIVs, also called candidate solution features, independent variables, or design variables.

As previously discussed, each archipelago corresponds to a subsystem. So each archipelago contains three groups of components. The first group of components is a group of islands, and each island is a possible solution to the subsystem optimization problem. The second group of components is a group of objectives for the subsystem. The last group of components is the set of constraints for the subsystem. The combination of all three groups of components in the subsystem is called an archipelago.

Mutation in BBO/Complex is identical to that in standard BBO. But migration in BBO/Complex needs to be modified due to the fact that the environment of BBO/Complex contains more than one subsystem. In the following subsections we consider two types of migration: within-subsystem migration and cross-subsystem migration.

4.3.1 Within-subsystem migration

In standard BBO, the fitness of an island is linearly related to the objective function because the system consists of only one objective function and no constraints. So the only performance measurement comes from the objective function. But in a complex system, the performance of an island is not reflected by only one objective function. That is the only difference between migration in BBO, and within-subsystem migration in BBO/Complex, but it is a major difference. Due to the fact that each subsystem contains multi-objectives and multi-constraints, we need to combine all of this information to determine the fitness of each island and its resulting migration rate.

We note here that Pareto-optimal solutions are often used in multi-objective algorithms [58]. But Pareto approaches require decision makers to select a single solution from a set of Pareto-optimal solutions, all of which are considered to be equally optimal. The Pareto approach has the advantage of providing multiple candidates to the decision maker as potential solutions, but has the drawback of requiring the decision maker to select from a potentially large set of such candidate solutions. Our approach avoids the need for a human decision maker, which may be desirable for certain problems.

In BBO/Complex, a modified version of the non-dominated ranking system (NDRS) [9] is used as the ranking system for islands. NDRS was initially designed for single systems with multi-objectives [59]. NDRS eliminates the weighting factors used in weighted ranking algorithms. NDRS can be easily deployed in almost any single-system, multi-objective optimization algorithm without major modification [60]. An updated version of NDRS was introduced in [61] as the ranking system in the multi-objective genetic algorithm (MOGA). That version uses non-consecutive integers as ranks to reflect the relative performance of each individual in a population. We are inspired here by both NDRS and the MOGA ranking system. But neither NDRS nor MOGA deals with constraint violation, which is a major concern in our work, as well as in most real-world optimization problems. So our modified NDRS considers constraint violations. We consider two factors that determine the relative performance of a candidate solution: fitness values and constraint violations. In our modified NDRS, the constraints have a higher priority than the fitness values. Violations of constraints significantly degrade the relative rank of individuals. Assume that we have a subsystem with the following characteristics: the population size is n; the number of objectives is m; the number of

constraints is k; R_i is the rank of the i-th island (to be determined below); and V_i is the number of constraint violations of the i-th island. Algorithm 1 outlines the modified NDRS procedure.

```
R_1 = R_2 =... = R_n = 0;
V_1 = V_2 =... = V_n = 0;
for i = 1 to n do
        for c = 1 to k do
                if constraint c of island i is violated then
                        V_i = V_i + 1
                end if
        end for
end for
for i_1 = 1 to n do
        for i_2 = i_1 to n do
                if V_i1 > V_i2
                        R_i1 = R_i1 + m
                else if V_i1 < V_i2
                        R_i2 = R_i2 + m
                else if V_i1 = V_i2
                        for o_1 = 1 to m do
                                if objective o_1 of island i_1 is better than o_1 of i_2 then
                                        R_i2 = R_i2 + 1
                                else if objective o_1 of island t_2 is better than o_1 of t_1
                                then
                                        R_i1 = R_i1 + 1
                                end if
                        end for
                end if
        end for
end for
```

Algorithm 1: Modified non-dominated ranking system (NDRS). V_i is the number of constraint violations of the i-th island, and R_i is the relative rank of the i-th island, where a lower rank is better. m is the number of optimization objectives.

After performing the above version of NDRS, we have the rank of each island in the subsystem. A smaller rank means better performance. For example, suppose have 4

islands and each of the islands has 3 objectives and 3 constraints. The objective and constraint violation information might be given in Table vi. Based on those data, the rank of each island is calculated according to the modified NDRS method in the last column of Table vi.

	Objective 1	Objective 2	Objective 3	Constraint Violation	Rank
Island 1	1	2	3	0	0
Island 2	2	4	2	1	4
Island 3	3	1	4	1	5
Island 4	1	1	1	2	9

Table vi: Rank calculation example with the modified NDRS. A lower objective means better performance, and lower ranks are better than higher ranks.

The ranks obtained from the modified NDRS are shown in Table vi, but one thing that needs to be mentioned is that the ranks assigned to the islands are 0, 4, 5, and 9. Ranks are not necessarily consecutive integers. The reason is that NDRS reflects the performance of an island by including the number of partial domination counts in a rank rather than simply ordering the islands. This gives more granularity for rank values, which is important when probabilistically choosing migrating islands in BBO.

4.3.2 Cross-subsystem migration

Standard BBO only contains one type of migration: within-subsystem migration, which has been modified for BBO/Complex as shown above. But BBO/Complex also includes cross-subsystem migration. Cross-subsystem migration is different because each subsystem has its own ranking system. The comparison of ranks across subsystems is

meaningless, because ranks assigned to each island in a subsystem only represents the relative goodness of the island in that specific subsystem. If we consider two islands in two different subsystems, we cannot determine which island is better by simply comparing their ranks, because ranks from different subsystems are calculated differently based on the different subsystem objectives and constraints. Instead, cross-subsystem migration is based on three factors – distance between islands, the similarity level of objectives, and the similarity level of constraints.

4.3.2.1 Distance between islands

The first factor to consider in cross-subsystem migration is the distance between islands. As we know, heuristic algorithms require population diversity [2]. BBO migration is based on sharing SIVs among islands. If the population has a low diversity, most of the islands are similar to each other, and the probability that an island improves after migration is low. In this case, migration may not effectively contribute to improvement in the population.

Mutation is the technique that introduces new SIVs to the population, and mutation does not depend on the diversity of the population. But the mutation rate is usually a small number, for example, 1%, because large mutation rates negate the effectiveness of migration and reduce the evolutionary algorithm to a random search. The new information introduced to the population through mutation sometimes includes useful SIVs. But most of the time, those SIVs are useless and can even degrade the population. In general, mutation is not a rapid or efficient technique for evolution.

Usually we use Euclidean distance to calculate the distance between islands. This calculation is straightforward for islands with the same structure. The Euclidean distance between island *a* and *b* in archipelago *h*, both of which have *c* SIVs, is

$$D_{hab} = \sqrt{\sum_{k=1}^{c} \left(S_{hak} - S_{hbk} \right)^2} \tag{4.1}$$

This calculation is valid if and only if both islands share the same structure, which means they have the same SIV type at the same location in the vector that defines the candidate solution. But in a complex system, subsystems usually have different island structures. That is, the independent variables in subsystems are not commensurate. For example, the SIV types in island 1 may be labeled type 1, 2 and 3; but the SIV types in island 2 may be labeled 2, 3, and 4. Equation (4.1) is not appropriate to calculate the distance between islands 1 and 2 in this case, because we cannot find the corresponding SIVs on both islands for the type 1 SIV and the type 4 SIV.

For BBO/Complex, we need a new technique to calculate the distance between islands with different structures. The partial distance strategy (PDS) is widely used in statistics to calculate Euclidean distances with missing data [62]. This is similar to our situation. Instead of missing data, we have missing SIV types. In order to implement PDS, we need to modify the data structure of the islands. First, we define each island to include all the SIV types on all islands, and this definition provides a unified format for islands. If an island did not originally include a specific SIV type, we assign an *N/A* value to the SIV and treat it as missing data. Assuming that there are a total of *t* types of SIVs, the unified format is given as follows:

$$x = \left[\mathrm{SIV}_1, \mathrm{SIV}_2, \mathrm{N/A}, ..., \mathrm{SIV}_t \right] \tag{4.2}$$

The implementation of PDS in BBO/Complex is given as follows.

$$D_{ghab} = \begin{cases} \dfrac{t}{K_{ghab}} \sqrt{\sum_{k=1}^{t} \left(S_{gak} - S_{hbk} \right)^2 K_{ghabk}}, & \text{if } K_{ghab} > 0 \\ 0, & \text{if } K_{ghab} = 0 \end{cases} \tag{4.3}$$

$$K_{ghabk} = \begin{cases} 0, & \text{if } S_{gak} = N / A \text{ or } S_{hbk} = N / A \\ 1, & \text{if } S_{gak} \neq N / A \text{ and } S_{hbk} \neq N / A \end{cases} \tag{4.4}$$

$$K_{ghab} = \sum_{k=1}^{t} K_{ghabk}$$

D_{ghab} is the partial distance between island a in archipelago g and island b in archipelago h; and t is the total number of SIV types. As an example, suppose we have 2 islands: island 1 = *[0 1 2 3, N/A, 4]*, and island 2 = *[1, 3, N/A, N/A, 5, 5]*. Island 1 has 5 SIVs, and island 2 has 4 SIVs, and the two islands have 3 SIVs in common. Then the distance is calculated based on Equation (4.3) and (4.4) as 4.90.

4.3.2.2 Similarities between objectives and constraints

The second and third factors in the island distance calculation are the similarity level of the objectives and the similarity level of the constraints. Subsystems with similar objectives and constraints are more likely to benefit each other through migration than subsystems that are not closely related. Our calculation of similarity level is based on the fast similarity level calculation (FSLC) [55]. Suppose there are two islands, each of which has a vector of independent variables: $U = [u_1, u_2, u_3, \ldots]$ and $V = [v_1, v_2, v_3, \ldots]$ (either objectives or constraints). The similarity level (SL) of these vectors is calculated by FSLC in Algorithm 2.

```
SL = 0
for each u ∈ U
        for each v ∈ V
                if u and v are the same type then
                        SL = SL + 1
                end if
        end for
end for
```

Algorithm 2: Similarity level calculation. U and V are the sets of objectives or constraints of two islands (candidate solutions).

4.3.2.3 Summary of cross-subsystem migration

Now that we have discussed the three factors for cross-subsystem migration, we summarize cross-subsystem migration as follows. First, calculate the migration probability between islands based on the similarity level between subsystems.

$$P_{migration} = \begin{cases} \frac{1}{2}\left(\frac{OS}{OS_{\max}} + \frac{CS}{CS_{\max}}\right), & \text{if } OS_{\max} > 0 \text{ and } CS_{\max} > 0 \\ \frac{1}{2}\frac{OS}{OS_{\max}}, & \text{if } OS_{\max} > 0 \text{ and } CS_{\max} = 0 \\ \frac{1}{2}\frac{CS}{CS_{\max}}, & \text{if } OS_{\max} = 0 \text{ and } CS_{\max} > 0 \\ 0, & \text{if } OS_{\max} = 0 \text{ and } CS_{\max} = 0 \end{cases} \tag{4.5}$$

OS is the objective similarity level between two islands; OS_{max} is the maximum inter-archipelago objective similarity level in the population; CS is the constraint similarity level between two islands; CS_{max} is the maximum inter-archipelago constraint similarity level in the population.

The probability for a pair of subsystems to perform cross-subsystem migration is linearly related to the above migration probability. After calculating the above probability, we need to choose emigrating islands for each immigrating island. We use roulette wheel selection [3] to select the emigrating island. Islands with better partial distances will have a better chance to be selected as the emigrating island. Figure 29 shows an example of emigrating island selection across subsystems.

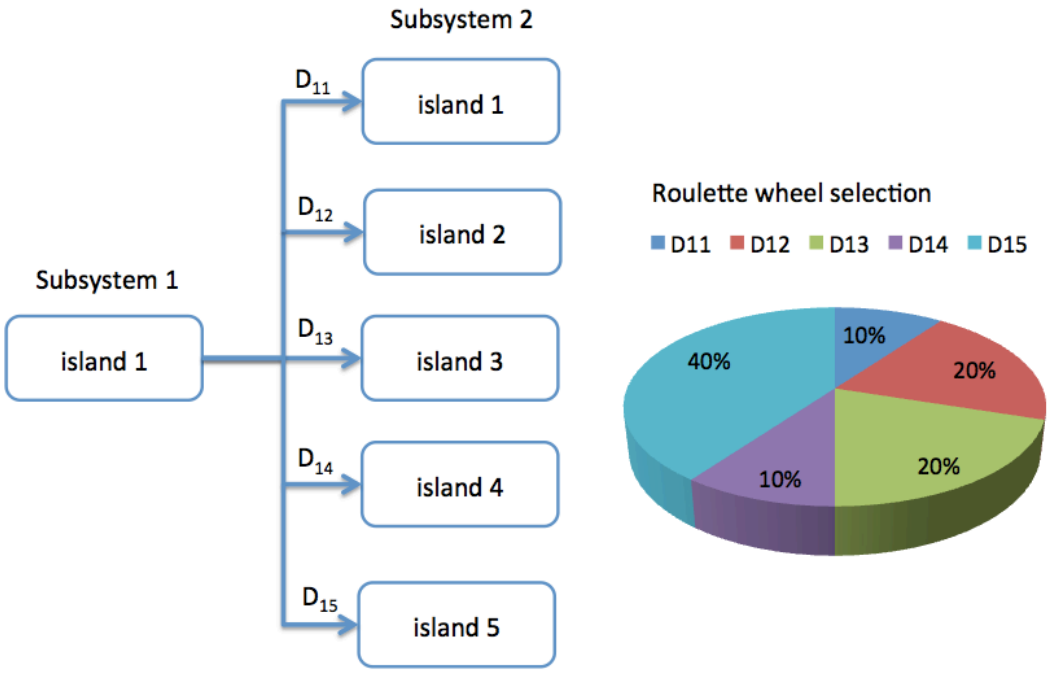

Figure 29: An example of emigrating island selection for immigration to island 1 in subsystem 1. First, calculate the partial distances between island 1 in subsystem 1, and each island in subsystem 2. Then create a roulette wheel based on the partial distances. Finally, probabilistically select the emigrating island based on roulette wheel selection.

4.3.3 Summary of BBO/Complex

BBO/Complex is summarized as follows.

1. Define the control parameters: population size, stopping criteria, mutation probability, and elitism parameter. A typical setup for BBO is that population size is 100, stopping criteria is 100,000 cost function calls, mutation probability is 0.05, and elitism parameter is 1.
2. Initialize the population. This is usually done with randomly-generated individuals.
3. Calculate the constraint and objective similarity levels between all pairs of subsystems.
4. Calculate the rank of islands in each subsystem.
5. Within-subsystem migration: Probabilistically choose the immigrating islands based on the island ranks. Use roulette wheel selection based on the emigration rates to select the emigrating islands. Emigration rates are linearly related to the island ranks. After each immigrating island selects its corresponding emigrating island, we perform within-subsystem migration. Each SIV in an immigrating island will have a chance to be replaced by an SIV from an emigrating island. This process is the same as migration in standard BBO.
6. Cross-subsystem migration: Find suitable pairs of subsystems based on similarity levels. Calculate distances between each pair of islands across all different subsystems. Use roulette wheel selection based on partial distances to select the emigrating islands. Then begin cross-subsystem migration. Each SIV in an

immigrating island will have a chance to be replaced by an SIV from an emigrating island, and this probability is $P_{SIVmigration}$ which can be predefined by users.

7. Probabilistically perform mutation on each island based on mutation probability.

8. In each subsystem, save the islands with the best performances as elite islands. Replace the worst islands in the population with the previous generation's elite islands.

9. If the termination criterion is not met, go to step 4; otherwise, terminate.

The structure of BBO/Complex is conceptually different than MDF, IDF, and CO. As we see from Figure 1, Figure 2, and Figure 3, MDF, IDF, and CO provide different strategies to optimize systems. But they are just frameworks, and we can choose any optimization method, like gradient descent or a GA, as the optimizer within the framework. But BBO/Complex is in a different category, because it includes both the framework and the optimization algorithm, as shown in Figure 30. It provides an efficient way to communicate between subsystems during the optimization process, and it provides a unique migration strategy to share information both within and across subsystems. Comparing MDF, IDF, CO, and BBO/Complex, we see that cross-subsystem migration in BBO/Complex is an innovation that can significantly enrich information sharing among subsystems.

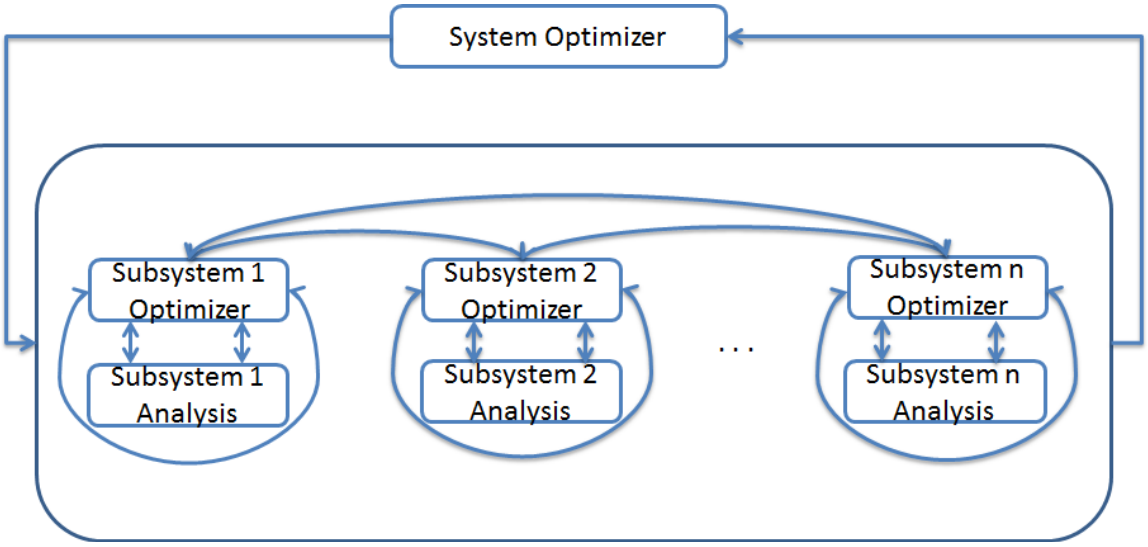

Figure 30: BBO/Complex formulation

4.4 Simulation

In this section, we compare the performance of BBO/Complex on real world benchmark problems with other well-known MDO algorithms: MDF, IDF, and CO. As we mentioned before, these three MDO algorithms are frameworks which require an additional optimization method as a complementary but essential component. The optimization algorithm we use in all three of these MDO algorithms is BBO without cross-subsystem migration. The benchmark problems are obtained from [62], and include the speed reducer problem, the propane combustion problem, the heart dipole problem, and the power converter problem. Each benchmark includes several subsystems, and each subsystem includes multi-objectives and multi-constraints. Detailed information about each benchmark can be found in the appendix.

The reason we choose these benchmarks is that they can be formulated as complex systems with inter-connected subsystems. There are two decomposition strategies: one is based on the physical system and one is based on the system requirements. In this section, we decompose the systems based on system requirements. Based on [64] and [65], traditional MDO algorithms usually lack the capability of dealing with multi-objectives, so their decomposition is based on the principle that each subsystem has one objective and multi-constraints. This type of decomposition is suitable for traditional optimization methods because it avoids the need to consider all objectives at once. Due to the fact that BBO is a heuristic algorithm, and with supporting results from [66] and [67], BBO/Complex is expected to perform well on multi-objective problems. It has more flexible decomposition options than traditional MDO algorithms.

Our decomposition option for BBO/Complex is that each subsystem has multi-objectives and multi-constraints. But in order to provide a fair comparison between other MDO algorithms and BBO/Complex, we also introduce a BBO/Complex version that uses the same decomposition strategy as the other MDO algorithms. So we have two versions of BBO/Complex in this section: the first one uses the same decomposition method as CO, MDF, and IDF, and is called BBO/Complex/Single; the other one uses multi-objectives in each subsystem, and is called BBO/Complex/Multi.

For each benchmark test, we compare the performance of each algorithm using both constraint violation and cost. We perform 100 Monte Carlo simulations for each algorithm and each benchmark problem to accurately measure performance. The termination criterion is 100,000 cost function evaluations. The constraint violation index is calculated for each generation as the average number of

constraint violations among all Monte Carlo simulations. They are all normalized between 0 and 1. The constraint violation index is 0 if there are no violations. The second performance metric is based on the cost function values. We calculate the rank for each algorithm using modified NDRS in each Monte Carlo simulation, and then obtain the averge rank over 100 Monte Carlo simulations. The optimization goal of each benchmark is to find the minimum value of the cost without violating any constraints. Since each benchmark contains multi-objectives, we use NDRS to calculate the rank of each algorithm based on its cost. But we have two priority levels: the first goal is to find feasible solutions, and the second goal is to reduce cost. Priority level one overrides priority level two.

4.4.1 The Speed Reducer Problem

The first benchmark we test is the speed reducer problem. It contains 3 objectives, 11 constraints, and 7 design variables, as detailed in the appendix. The performance of all algorithms on the first benchmark is shown in Table vii, which shows that BBO/Complex/Single has the best performance on the speed reducer benchmark, including the best cost rank and the minimum constraint violation. MDF, CO, and BBO/Complex/Multi are slightly worse than BBO/Complex/Single. IDF has the worst performance in terms of both cost rank and constraint violation.

	NDRS Cost Rank	SD	Violation
BBOComplex/Single	2.41	2.06	0.04
MDF	2.62	2.11	0.05
CO	5.17	2.10	0.08
BBOComplex/Multi	7.80	1.59	0.14
IDF	12.00	0	0.27

Table vii: NDRS cost ranks, standard deviation of ranks, and constraint violations for the speed reducer problem after 100,000 function calls. For each metric, a smaller number means better performance.

4.4.2 The Power Converter Problem

The second benchmark is the power converter problem. It has 6 design variables, 8 state variables, 2 objectives, and 4 constraints, as detailed in the appendix. Table viii shows the performance of the algorithms on the power converter problem. The performances of all algorithms are fairly close to each other. We have good results on this problem because all algorithms achieve a 0 constraint violation. CO is the best algorithm in terms of cost, and BBO/Complex/Multi has the second best performance.

	NDRS Cost Rank	SD	Violation
CO	3.51	0.56	0
BBOComplex/Multi	3.73	0.47	0
MDF	3.76	0.57	0
BBOComplex/Single	3.77	0.57	0
IDF	5.23	1.35	0

Table viii: NDRS cost ranks, standard deviation of ranks, and constraint violations for the power converter problem after 100,000 function calls. For each metric, a smaller number means better performance.

4.4.3 The Heart Dipole Problem

The third benchmark is the heart dipole problem. It has 6 design variables, 2 objectives, and 5 constraints, as detailed in the appendix. Table ix shows that BBO/Complex/Single, BBO/Complex/Multi and MDF achieve a 0 constraint violation, which means that the best individuals for each Monte Carlo run are feasible. When we combine cost rank and constraint violation, BBO/Complex/Single has the best performance on this benchmark, and BBO/Complex/Multi is the second best.

	NDRS Cost Rank	SD	Violation
BBOComplex/Single	1.35	1.30	0
BBOComplex/Multi	1.36	1.16	0
MDF	3.29	1.17	0
IDF	6	0	0.20
CO	8	0	0.40

Table ix: NDRS cost ranks, standard deviation of ranks, and constraint violations for the heart dipole problem after 100,000 function calls. For each metric, a smaller number means better performance.

4.4.4 The Propane Combustion Problem

The fourth benchmark is the propane combustion problem. It has 1 design variable, 3 objectives, and 4 constraints, as detailed in the appendix. According to Table x, BBO/Complex/Multi is the best algorithm for this benchmark because it is the only algorithm that achieves a 0 constraint violation. BBO/Complex/Single achieves the second best performance with a constraint violation slightly greater than 0.

	NDRS Cost Rank	SD	Violation
BBOComplex/Multi	1.71	1.58	0
BBOComplex/Single	2.99	2.06	0.02
CO	4.75	1.99	0.04
MDF	9.76	0.46	0.25
IDF	10.79	0.49	0.25

Table x: NDRS cost ranks, standard deviation of ranks, and constraint violations for the propane combustion problem after 100,000 function calls. For each metric, a smaller number means better performance.

4.5 Summary of Benchmark Tests

The benchmark results show that BBO/Complex/Multi is the only algorithm that obtains feasible solutions on three of the benchmarks. For the speed reducer benchmark, none of the algorithms finds a feasible solution, but BBO/Complex/Single comes the closest. Among all four benchmarks, BBO/Complex/Multi achieves the best performance once and the second best performance twice, and BBO/Complex/Single achieves the best performance twice and the second best performance once. Among the non-BBO/Complex algorithms, CO is the best, achieving the best performance once.

4.6 Markov model of BBO/Complex

As typified by BBO algorithm described earlier, most heuristic algorithms have a similar evolution process in their search for an optimal solution. In contrast with more traditional and analytic optimization algorithms, there is no guarantee that we can obtain the optimal solution with heuristic algorithms.

Markov models are general tools that are used to describe the probability of transitioning from one state to another. If we can develop a Markov model for a system, the probability of the appearance of each state can be calculated mathematically. If we treat a Markov state as a distribution of individuals in a heuristic algorithm, then we can use the Markov model to calculate the probability of the appearance of any given population distribution, which means that we can calculate the probability that the optimal solution will be found by the heuristic algorithm. In this way, Markov models can be used to mathematically analyze the performance of heuristic algorithms for given optimization problems. Markov models have been successfully applied to various heuristic algorithms, such as simple genetic algorithms [68], simulated annealing [69], the genitor algorithm, and the CHC algorithm [70]. In 2010, a Markov model was developed for BBO, and that was the first time that the performance of BBO was analyzed mathematically and theoretically [7]. The following sections extend the BBO Markov model to the BBO/Complex algorithm.

4.6.1 Development of a Markov model of BBO/Complex

Markov models describe the probability that a system transitions from one state to another. They are discrete-time random processes on a finite state space. Assume that there are T possible states in some system. Then a $T \times T$ transition matrix can be defined to describe the probability of transitioning between each pair of states. We call this transition matrix P. The probability that state S_i transits to state S_j is given by P_{ij}, which is also called the transition probability. If a Markov model can transition from any state to any other state, then P does not include any zero entries, and the transition matrix P of the Markov chain is called regular. If P is regular, we can obtain the steady state transition matrix P_{ss} as follows [7], [71]:

$$\lim_{n \to \infty} P^n = P_{ss} \tag{4.6}$$

Equation (4.6) gives the transition matrix after an infinite number of transitions. Each row in P_{ss} is the same as every other row, and the i-th element in each row is the limiting probability of the occurrence of state i as the number of transitions approaches infinity.

If BBO/Complex is implemented on a system with discrete independent variables, then it has a finite number of population distributions, and we can derive a Markov model for it. Each population distribution represents a state in the Markov model. As shown in Equation (4.6), P_{ss} is independent from the initial state. In BBO, this means that the final population distribution is independent of the initial population. This result is of great importance in building a Markov model for BBO/Complex. We only need the transition matrix to predict the final population distribution (in the limit as the generation count

approaches infinity), and this limiting distribution is independent from the initial population. For the simulations that will be used to verify the Markov model later, we do not need to be particular about the initial population – all initial populations will eventually lead to the same final population distribution.

The environment of BBO/Complex is comprised of M subsystems. We assume here that the independent variables of the optimization problem are binary, although we note that any discrete space can easily be mapped into a binary space. The number of bits in each island (candidate solution) in subsystem i is denoted as b_i. The population size of BBO for subsystem i is denoted as n_i. The total number of possible solutions in subsystem i is denoted as N_i, and the total number of possible solutions in the entire system is denoted as N. N_i and N are calculated as follows:

$$\begin{aligned} N_i &= 2^{b_i} \\ N &= \prod_{i=1}^{M} N_i \end{aligned} \tag{4.7}$$

The j-th island (candidate solution) in the population of subsystem i is denoted as y_{ij}. The j-th point in the search space of the subsystem i is denoted as x_{ij}. We use v_{ij} to denote the total number of x_{ij} islands in subsystem i. So the combined BBO/Complex population can be generally represented as follows:

$$\begin{aligned}\text{Population} &= \left\{\left[y_{11},\ldots,y_{1N_1}\right],\left[y_{21},\ldots,y_{2N_2}\right],\ldots,\left[y_{M1},\ldots,y_{MN_M}\right]\right\} \\ &= \left\{\left[\underbrace{x_{11},\ldots,x_{11}}_{v_{11}},\underbrace{x_{12},\ldots,x_{12}}_{v_{12}},\ldots,\underbrace{x_{1N_1},\ldots,x_{1N_1}}_{v_{1N_1}}\right],\right. \\ &\quad \left[\underbrace{x_{21},\ldots,x_{21}}_{v_{21}},\underbrace{x_{22},\ldots,x_{22}}_{v_{22}},\ldots,\underbrace{x_{2N_2},\ldots,x_{2N_2}}_{v_{2N_2}}\right], \\ &\quad \vdots \\ &\quad \left.\left[\underbrace{x_{M1},\ldots,x_{M1}}_{v_{M1}},\underbrace{x_{M2},\ldots,x_{M2}}_{v_{M2}},\ldots,\underbrace{x_{MN_M},\ldots,x_{MN_M}}_{v_{MN_M}}\right]\right\}\end{aligned} \tag{4.8}$$

For convenience in notation, we have ordered the y_{ij} islands in the same order as the x_{ij} search space points. Based on Equation (4.8), the population in subsystem i can be written in a more compact format as follows:

$$y_{ik} = \begin{cases} x_{i1}, \text{ when } k = 1,...,v_{i1} \\ x_{i2}, \text{ when } k = v_{i1}+1,...,v_{i1}+v_{i2} \\ \vdots \\ x_{iN_i}, \text{ when } k = \sum_{l=1}^{N_i-1} v_{il}+1,...,\sum_{l=1}^{N_i} v_{il} \end{cases} \tag{4.9}$$

for i = 1, …, M. In Equation (4.9), y_{ik} denotes the k-th island in the population of subsystem i. We use $y_{ik}(s)$ to represent the s-th bit in the k-th island in the population of subsystem i. Equation (4.9) can be written as follows:

$$\begin{aligned} y_{ik} &= x_{iz(k)} \\ z(k) &= \min r, \text{ such that } \sum_{l=1}^{r} v_{il} > k \end{aligned} \tag{4.10}$$

Based on the definition of BBO/Complex, islands in different subsystems have different structures and contain different types of SIVs. For ease of notation, we use a

unified format to represent each island in the BBO/Complex population, as shown in Equation (4.2).

A Markov model describes the transitions between states. Each state in BBO/Complex is a specific population distribution. Each generation of BBO/Complex updates its population with migration and mutation. A transition between states corresponds to the evolution of the population in one generation of BBO/Complex. So in order to build a transition matrix, we need to model migration and mutation in BBO/Complex. In the following subsections, we study the migration and mutation processes, and use them to build the transition matrix for the Markov model of BBO/Complex.

4.6.1.1 Migration

Migration is the main technique that BBO uses to share information among islands. In the original BBO algorithm, the basic procedure of migration is to probabilistically select an immigrating island (an island that imports SIVs) and an emigrating island (an island that exports SIVs). Then we probabilistically choose some SIVs from the emigrating island, and use them to replace the SIVs in the immigrating island. Figure 31 illustrates a simple migration process.

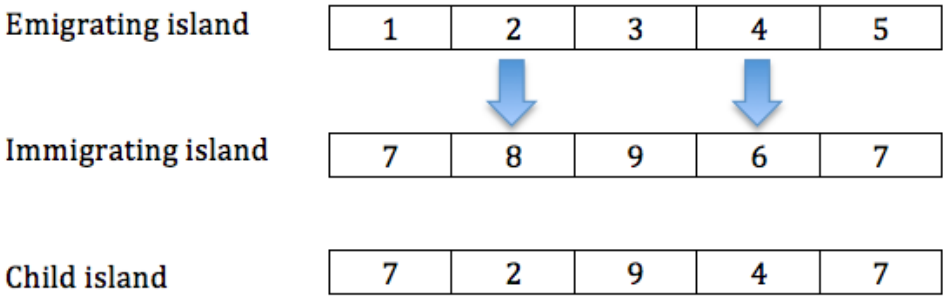

Figure 31: An example of migration between two islands.

In BBO/Complex, the migration process is more complicated. Rather than having just one population, it contains multiple populations, one for each subsystem. Each subsystem is relatively independent from the others, which means the construction of the population of each subsystem is relatively independent from the others. Also, there are two types of migration in BBO/Complex, within-subsystem migration and cross-subsystem migration, which introduces further complexity to the Markov model development.

There are four assumptions we make in this section to develop the Markov model. They are similar (but expanded) versions of the assumptions used to develop the Markov model for the original BBO algorithm [7].

First, a BBO solution will not be replaced until the end of the generation. In other words, BBO is generational rather than steady-state [2]. This assumption guarantees that the migration probabilities remain the same throughout each generation.

Second, an island can emigrate to itself. The immigrating and emigrating islands are probabilistically chosen from the entire population. So there is a chance that the

immigrating and emigrating islands are the same. This is similar to a chromosome crossing over with itself in a GA.

Third, migration only happens between SIVs with the same type. The environment of BBO/Complex is a group of archipelagos. Each archipelago has a unique population structure, depending on the subsystem with which it is associated. So islands from different archipelagos might not contain the same SIV types. SIVs represent features, and each feature has a unique domain. That is the reason that migration is only valid between the same SIV types. For example, suppose some island consists of five SIVs, where SIV1 is the proportional gain of a PID controller, SIV2 is the integral gain of a PID controller, and SIV3 is the derivative gain of a PID controller. Each SIV has a unique type, definition, and parameter domain. For example, the domain of SIV1 might be from 0.5 to 1, while the domain of SIV2 might be from 0.1 to 0.4. Migration between SIV1 and SIV2 would not make sense because SIV1 represents a proportional gain wile SIV2 represents an integral gain, which is a completely different parameter with a completely different function.

Fourth, we use predetermined migration rates for each island rank rather than calculating the migration rates each generation. All the ranks are calculated each generation based on the non-dominated sorting method [9]. The emigration rate μ and immigration rate λ of each island are calculated based on the rank of the island, which is similar to the original BBO algorithm, except here we use ranks based on multi-objectives and multi-constraints, instead of ranks based on scalar cost values.

Within-Subsystem Migration

BBO/Complex contains two types of migration: within-subsystem migration and cross-subsystem migration. Within-subsystem migration is similar to the original migration method in BBO, and it is used for migration between islands within the same subsystem. This migration process has two possible situations. First, since migration is selected probabilistically, it might not be performed, which means the features in the potential immigrating island will not be changed from one generation to the next. This situation is represented for the *k*-th individual in the *i*-th subsystem as follows:

$$y_{ik}(s)_{t+1} = y_{ik}(s)_t = x_{iz(k)}(s) \tag{4.11}$$

The second situation is that a feature is selected to migrate to the immigrating island. The probability of obtaining a certain bit at a certain locus in a given island is proportional to two factors: the total number of occurrences of that bit in the entire subsystem population; and the emigration rates of the islands that contain those bits. This probability is calculated for the *k*-th individual in the *i*-th subsystem as follows:

$$\Pr(y_{ik}(s)_{t+1} = x_{il}(s) \mid \text{immigration}) = \frac{\sum_{j \in J_{il}(s)} v_{ij}\mu_{ij}}{\sum_{j=1}^{n_i} v_{ij}\mu_{ij}} \tag{4.12}$$

where $J_{il}(s)$ is the set of islands in subsystem *i* that contain the same bit in position *s* as island x_{il}:

$$J_{il}(s) = \left\{ j : x_{ij}(s) = x_{il}(s) \right\} \tag{4.13}$$

Considering both situations described above and combining them into one equation, we obtain the probability of obtaining a given bit from within-subsystem migration:

$$\begin{aligned}
&\Pr_{im-within-sub}(y_{ik}(s)_{t+1} = x_{il}(s)) \\
&= \Pr(\text{no immigration})(y_{ik}(s)_{t+1} = x_{il}(s) \,|\, \text{no immigration}) \\
&\quad + \Pr(\text{immigration})(y_{ik}(s)_{t+1} = x_{il}(s) \,|\, \text{immigration}) \\
&= (1-\lambda_{iz(k)})1_0(x_{iz(k)}(s) - x_{il}(s)) + \lambda_{iz(k)} \frac{\sum_{j\in J_{il}(s)} v_{ij}\mu_{ij}}{\sum_{j=1}^{n_i} v_{ij}\mu_{ij}}
\end{aligned} \tag{4.14}$$

Assume we have q bits in our unified island format, as given in Equation (4.2). The probability that the k-th individual in the i-th subsystem is equal to a given island x_{il} can be calculated based on Equation (4.14), which shows the probability of obtaining a single bit. We use $P_{ikl}^{(1)}(v)$ to denote this probability, which is a function of the current population vector v at the t-th generation. (The term *population vector* will be defined later, but for now we simply need to know that it represents the current population in the BBO algorithm.) This probability is given as follows:

$$\begin{aligned}
P_{ikl}^{(1)}(v) &= \Pr(y_{ik,t+1} = x_{il}) \\
&= \prod_{s=1}^{q} \left[(1-\lambda_{iz(k)})1_0(x_{iz(k)}(s) - x_{il}(s)) + \lambda_{iz(k)} \frac{\sum_{j\in J_{il}(s)} v_{ij}\mu_{ij}}{\sum_{j=1}^{n_i} v_{ij}\mu_{ij}} \right]
\end{aligned} \tag{4.15}$$

Cross-Subsystem Migration

The second type of migration is called cross-subsystem migration, which is the migration process between subsystems. Cross-subsystem migration is more complicated

than within-subsystem migration for two reasons: (1) the island structure varies from subsystem to subsystem; (2) ranking cannot be used to compare the performance cost of islands across subsystems, because rank information is useless when islands do not share identical cost and constraint functions. Issue one can be addressed with our unified island structure, which is shown in Equation (4.2). For issue two, we need to introduce a new strategy for island selection that is specifically geared toward the optimization of multiple related subsystems, and this strategy will be introduced later in this section.

For cross-subsystem migration, when considering the possibility of immigration to a given individual, we have the same two possibilities as we do for within-subsystem migration: (1) migration is not performed (recall that the migration decision is made probabilistically); (2) migration is performed. The first scenario is exactly like the corresponding scenario in within-subsystem migration. The only thing we need to reconsider here is the second possibility, and how to compute the probability of occurrence of each island after migration, since ranks within a subsystem do not indicate their cost values relative to islands in other subsystems. BBO/Complex introduces the concept of distances between islands for the selection of islands in cross-subsystem migration [62], [28]. The motivation of this method is based on the concept of diversity: a larger diversity in a population provides more opportunities to find an optimal solution. The probability that we obtain a given bit $x_{il}(s)$ at a given position s in the k-th individual y_{ik} in the i-th subsystem is calculated as follows:

$$\begin{aligned}
&\Pr_{im-cross-sub}(y_{ik}(s)_{t+1} = x_{il}(s) \mid \text{immigration from subsystem } m) \\
&= \Pr(\text{no immigration})(y_{ik}(s)_{t+1} = x_{il}(s) \mid \text{no immigration}) \\
&\quad + \Pr(\text{immigration})(y_{ik}(s)_{t+1} = x_{il}(s) \mid \text{immigration}) \\
&= (1-\lambda_{iz(k)})1_0(x_{iz(k)}(s) - x_{il}(s)) + \lambda_{iz(k)} \frac{\sum_{j \in J_{il}(s)} v_{mj}\sigma_{ilmj}}{\sum_{j=1}^{n_m} v_{mj}\sigma_{ilmj}}
\end{aligned} \tag{4.16}$$

σ_{ilmj}: distance between island l in subsystem i and island j in subsystem m.

The probability that $y_{ik,t+1}=x_{il}$ after cross-subsystem migration can be calculated based on Equation (4.16), and is denoted as $P_{ikl}^{(2)}(v)$.

$$\begin{aligned}
P_{ikl}^{(2)}(v) &= \Pr(y_{ik,t+1} = x_{il}) \\
&= \prod_{s=1}^{q} \left[(1-\lambda_{iz(k)})1_0(x_{iz(k)}(s) - x_{il}(s)) + \lambda_{iz(k)} \frac{\sum_{j \in J_{il}(s)} v_{mj}\sigma_{ilmj}}{\sum_{j=1}^{n_m} v_{mj}\sigma_{ilmj}} \right]
\end{aligned} \tag{4.17}$$

Combined Within-Subsystem Migration and Cross-Subsystem Migration

Recall that the beginning of this chapter provides the detailed BBO/Complex procedure. According to this procedure, there are three steps in modifying a population, with the sequence given as follows: within-subsystem migration, cross-subsystem migration, and mutation. To find the total probability of obtaining a given island, we need to combine the probabilities of those three processes. Based on our derivations up to this point, we combine the probabilities of the two types of migration, and we use $P_{ikl}^{(3)}(v)$ to denote this probability.

$$P_{ikl}^{(3)}(v) = \sum_{j=1}^{n_i} P_{ikj}^{(1)}(v) P_{ijl}^{(2)}(v) \tag{4.18}$$

This is the probability that y_{ik} is equal to x_{il} after both within-subsystem and cross-subsystem migration have been considered.

4.6.1.2 Mutation

Mutation is another way to alter islands in BBO/Complex. In order to calculate the probability of obtaining a certain island after two migrations and one mutation, we need to follow two steps. First, we obtain the probability of transforming a given island to another given island due to mutation, and then we combine this probability with Equation (4.18) to obtain the total probability.

Assuming that the mutation rate is predefined and constant, we can easily create a mutation matrix for each subsystem. When we denote the mutation matrix as U_i for subsystem i, the mutation probability that island x_{ir} mutates to island x_{il} is represented by U_{irl} which is the l-th element in the r-th row in the mutation matrix U_i. So the size of U_i is $n_i \times n_i$. We next combine U_i with Equation (4.18) to obtain the probability that $y_{ik,t+1}=x_{il}$ after within-subsystem migration, cross-subsystem migration, and mutation have all been considered:

$$P_{ikl}^{(4)}(v) = \sum_{r=1}^{n_i} \sum_{j=1}^{n_i} P_{ikj}^{(1)}(v) P_{ijr}^{(2)}(v) U_{irl} \tag{4.19}$$

Now we will extend the probability from the island level to the population level. Before we do that, there is a term that needs to be introduced – *population vector*. In BBO/Complex, the population distribution is represented by a population vector. This is

best illustrated by example. Assume we have two subsystems with four possible islands in subsystem 1, and four possible islands in subsystem 2. Then the population vector contains eight elements as illustrated in Figure 32.

Population vector							
Island-11 Count	Island-12 Count	Island-13 Count	Island-14 Count	Island-21 Count	Island-22 Count	Island-23 Count	Island-24 Count

Figure 32: Population vector for a system that is comprised of two subsystems, where each subsystem has a search space cardinality of four. The population vector has eight elements. Island-*ik* represents the number of x_{ik} individuals in subsystem *k*.

As an example based on Figure 32, a population vector [0 0 2 2 3 1 0 0] indicates that subsystem 1 contains two island-3 individuals, and 2 island-4 individuals; and subsystem 2 has three island-1 individuals and one island-2 individual.

We follow the method from [7], [72], and use the generalized multinomial theorem to find the probability that population vector *v* transitions to population vector *u* in subsystem *i* after one generation, and we use $\Pr_i(u|v)$ to denote this probability:

$$\Pr_i(u \mid v) = \sum_{J \in Y_i} \prod_{k=1}^{N_i} \prod_{l=1}^{n_i} \left(P_{ikl}^{(4)}(v) \right)^{J_{ikl}}$$
$$Y_i = \left\{ J_i \in R^{N_i \times n_i} : J_{ikl} \in \{0,1\},\ \sum_{l=1}^{n_i} J_{ikl} = 1 \text{ for all } k,\ \sum_{k=1}^{N_i} J_{ikl} = u_{il} \text{ for all } l \right\} \tag{4.20}$$

Based on Equation (4.20), we obtain the transition matrix P_i for subsystem *i*. Each element in P_i represents the probability of transitioning from one possible population

vector to another. P_i is a $T_i \times T_i$ matrix, where T_i is the total number of possible population vectors in subsystem i. T_i can be calculated as follows [7]:

$$T_i = \begin{pmatrix} n_i + N_i - 1 \\ N_i \end{pmatrix} \tag{4.21}$$

Note that there are $T_i \times T_i$ combinations of u and v vectors in Equation (4.20). These $T_i \times T_i$ different probabilities comprise the entries of the P_i transition matrix. After obtaining the transition matrices of each subsystem, we combine the matrices to form the transition matrix P for the entire system. The size of P is $T \times T$, where T is the total number of possible population vectors for the entire system:

$$T = \prod_{i=1}^{M} T_i \tag{4.22}$$

where M is the number of subsystems in the entire complex system. The P matrix can be calculated in pseudo-code as shown in Algorithm 3.

```
FOR (z1= 0; z1++; z1<t) {
    SET Count = 0;
    FOR (z2=0; z2++; z2<t1) {
        FOR (z3=0; z3++; z3<t2) {
                ⋮

            FOR (zM+1=0; zM+1++; zM+1<tM) {
                P(Count, z1) = P1(z2,z1)P2(z3,z1)...Py(zM+1,z1);
                Count++;
            }
        }
    }
}
```

M: number of subsystems

t: number of possible population distributions for entire system

t_i: number of possible population distributions for subsystem *i*

$P_i(i,j)$: element in *i*-th row and *j*-th column of the transition matrix of subsystem *i*

Algorithm 3: Pseudo-code to construct *P* matrix

After calculating transition matrix *P*, the probability of each possible population, in the limit as the generation count approaches infinity, can be calculated based on Equation (4.6).

4.6.3 Simulation

In the first part of this chapter, we introduced a method to calculate the limiting probability of each possible population, which exactly predicts the steady state probability of each population vector during BBO/Complex. Now, we use a sample problem to confirm the newly derived Markov model.

The sample problem is a complex system which has two subsystems. Each subsystem contains two bits. Subsystem 1 contains type-1 and type-2 bits; subsystem 2 contains type-1 and type-3 bits. The subsystems share type-1 bits in common. The possible islands of subsystem 1 and subsystem 2 in a unified format are shown in Table xi and Table xii.

	Type-1 bit	Type-2 bit	Type-3 bit
Possible island 1	0	0	N/A
Possible island 2	0	1	N/A
Possible island 3	1	0	N/A
Possible island 4	1	1	N/A

Table xi: Possible islands of subsystem 1

	Type-1 bit	Type-2 bit	Type-3 bit
Possible island 1	0	N/A	0
Possible island 2	0	N/A	1
Possible island 3	1	N/A	0
Possible island 4	1	N/A	1

Table xii: Possible islands of subsystem 2

Each subsystem includes two cost functions. A smaller cost means better performance. The cost functions for subsystem 1 are given as follows:

$$\begin{aligned} y_{11} &= 2x_{11} + x_{12} + 1 \\ y_{12} &= \frac{y_{11}}{x_{11} + x_{12} + 1} + 1 \end{aligned} \tag{4.23}$$

- y_{11}: first cost value of an island in subsystem 1
- y_{12}: second cost value of an island in subsystem 1
- x_{11}: first bit of an island in subsystem 1
- x_{12}: second bit of an island in subsystem 1

The cost functions for subsystem 2 are given as follows:

$$\begin{aligned} y_{21} &= 2x_{21} + x_{23} + 1 \\ y_{22} &= \frac{x_{21} + x_{23} + 1}{y_{21} + 1} + 1 \end{aligned} \tag{4.24}$$

- y_{21}: first cost value of an island in subsystem 2
- y_{22}: second cost value of an island in subsystem 2
- x_{21}: first bit of an island in subsystem 2
- x_{22}: second bit of an island in subsystem 2

In order to verify the BBO/Complex Markov model derived in the previous section, we have two requirements for the simulation setup. First, we need to perform Monte Carlo simulations of BBO/Complex to obtain average performance. Second, the generation limit of each BBO/Complex Monte Carlo simulation should be large enough that the simulation results converge to steady state values. These number of Monte Carlo simulations, and the number of generations of each simulation, are determined empirically. The simulation setup is shown as follows.

- Monte Carlo simulations: 100
- BBO/Complex generations for each Monte Carlo simulation: 5000

- Number of subsystems: 2
- Number of islands per subsystem (population size): 4
- Number of bits per island: 3

We optimize this sample problem with three different mutation rates in BBO/Complex: 0.001, 0.01, and 0.1.

Mutation Rate	Population Vector	Probability	
		Markov	Simulation
0.001	4 0 0 0 4 0 0 0	0.9489	0.9590
	3 1 0 0 4 0 0 0	0.0287	0.0194
	4 0 0 0 3 1 0 0	0.0105	0.0074
	4 0 0 0 3 0 1 0	0.0060	0.0070
	3 0 1 0 4 0 0 0	0.0044	0.0058
0.01	4 0 0 0 4 0 0 0	0.6051	0.5901
	3 1 0 0 4 0 0 0	0.1655	0.1770
	4 0 0 0 3 1 0 0	0.0647	0.0631
	4 0 0 0 3 0 1 0	0.0385	0.0425
	3 0 1 0 4 0 0 0	0.0284	0.0294
0.1	3 1 0 0 4 0 0 0	0.0425	0.0348
	3 1 0 0 3 1 0 0	0.0371	0.0268
	2 2 0 0 4 0 0 0	0.0329	0.0274
	2 2 0 0 3 1 0 0	0.0287	0.0218
	3 1 0 0 3 0 1 0	0.0278	0.0258

Table xiii: The five most likely populations for three mutation rates.

Based on the cost functions for each subsystem and the non-dominated ranking system, the optimal population vector is [4 0 0 0 4 0 0 0]. According to the results shown in Table xiii, when the mutation rate is 0.001, the probability of obtaining the optimal

population vector calculated by the Markov model is 0.9489, and the probability calculated by the simulation is 0.9590. This confirms that the optimal population vector dominates other populations, and we have a high probability of obtaining it. Also, the simulation results match the theoretical results well.

When the mutation rate is 0.01, the most probable population vector is still the optimal one, but the probability of the optimal population vector falls to around 60%. Although performance is degraded, the probabilities calculated by the Markov model and by the simulation are still close.

When the mutation rate is 0.1, the most probable population vector is [3 1 0 0 4 0 0 0], which is not the optimal population vector. The optimal population vector [4 0 0 0 4 0 0 0] is only the 7th most likely according to the Markov model, and the 5th most likely according to the simulation (not shown in Table xiii). Since the probability values are relatively small for the top five population vectors, the differences between the Markov model results and the simulation results are larger compared to when the mutation rate is lower, but the differences between theory and simulation are still small. The theoretical Markov model results are exactly correction, but the simulation results are only approximate due to the stochastic nature of the BBO/Complex algorithm.

Based on Table xiii, the Markov model is verified by the simulation results. Finally, note that the calculation time for the Markov model probabilities was 492 seconds, but the average calculation time of each set of Monte Carlo simulations was 1166 seconds. In this case, the Markov model not only obtained more accurate steady-state results than the simulation, but also did so with less computational time.

CHAPTER V

CONCLUSION AND FUTURE WORK

5.1 Conclusion

In Chapter 2, the focus was on an efficiency test and convergence analysis for heuristic algorithms. Heuristic algorithms are often implemented on large systems with complex structures. Analytical optimization of these systems is hard to achieve. In addition, optimization based on heuristic algorithms is time consuming, and the quality of the final result is not guaranteed. Flexibility is one of the main benefits of heuristic algorithms, but heuristic algorithms have drawbacks. Flexibility allows us to implement heuristic algorithms without knowing the details of the problem, but it also results in slow convergence. In Chapter 2, we tested two aspects of BBO: the initial population construction and the information sharing process. In both aspects, use of problem specific characteristics can have a large effect. Specially modified algorithms clearly outperform algorithms without any modification. Also, we conducted a convergence analysis based

on BBO, and it showed that the mutation technique can guarantee that BBO will eventually find the optimal solution, which makes BBO a true global optimization method.

In Chapter 3, we introduced BBO for combinatorial problems. Since TSP is a representative example of combinatorial problems, all the simulation examples in Chapter 3 were TSPs. First, based on the results from Chapter 2, we saw that a good population initialization method can result in significant differences in performance. We introduced a population initialization method, NNA, into BBO. Based on the simulation results, we saw that it provided a big improvement compared to standard BBO. Second, crossover methods specially designed for TSP were introduced to BBO. When combining BBO with other crossover techniques like matrix crossover, cycle crossover and inver-over crossover, BBO becomes compatible with combinatorial problems. Third, local optimization methods were introduced into BBO. As we know, the information sharing strategies of most heuristic algorithms are designed for global optimization. The advantage of this type of design is that heuristic algorithms can search for the globally optimal solution, and not get easily stuck in locally optimal solutions. In contrast, local optimization methods are designed for seeking locally optimal solutions. Since the domain of the local search area is fairly small, the search process is much faster than a global search. For combinatorial problems, each possible solution contains all the necessary information to construct an optimal solution. When combining the power of both global optimization and local optimization, we improved the performance of BBO. The simulation results also confirmed this. The last technique we introduced into BBO was greedy methods, and it showed its potential on large problems. In the end, a modified

BBO was created which benefits from the previous studies by combining the techniques with the best performance: 1 NNA for population initialization; inver-over crossover; k-opt for local optimization; and all greedy for the greedy method. The modified BBO obtained promising simulation results when compared with other well-known algorithms. At the end of Chapter 3, a TSP GUI was built based on BBO. This GUI contains all the compatible TSPs from TSPLib, and provides a user-friendly interface to let users intuitively explore the different techniques in Chapter 3. This GUI is not only a test platform, but also a modularized BBO implementation with a well-designed interface between the main BBO algorithm and the other modules, including population initialization, migration, mutation, etc. Users can easily build their only BBO algorithms with other techniques of their choice with this GUI. This GUI can be a useful tool for both teaching and researching.

In Chapter 4, a new topic was addressed: BBO for complex systems. Systems built in recent years are more complicated than ever, and complex systems have become quite common these days. The aim of traditional heuristic algorithms is usually to optimize one system. Complex systems have three major challenges: multi-objectives, multi-constraints, and multi-subsystems. The last challenge, multi-subsystems, has not been widely addressed before now in evolutionary optimization research. In Chapter 4, a newly designed BBO algorithm called BBO/complex was introduced. Based on the new immigration probability calculation method and the ranking method, we successfully created a BBO algorithm for complex systems. BBO/Complex uses the original framework of standard BBO, but extends it to a multi-archipelago environment to suit the structure of complex systems. BBO/Complex has one significant difference from its

predecessors – it combines the optimization framework and the low-level optimization approach into a single algorithm. This is quite different from MDF, IDF, and CO, all of which are only frameworks for complex system optimizers, and which need a low-level optimization method as an additional tuning parameter. The low-level optimization approaches incorporated in MDF, IDF, and CO are typically traditional algorithms like gradient descent, Newton's method, etc. But those algorithms can easily get stuck in a local optimum. Based on [7] and [73], standard BBO can guarantee convergence to the optimal solution given enough generations. Besides the traditional advantages of BBO, the BBO/Complex algorithm also introduces new features, like a ranking system that evaluates candidate solutions based on both performance and constraints, the use of PDS to maintain the diversity of the population, within-subsystem migration for information sharing within subpopulations, and cross-subsystem migration for information sharing between subpopulations. The simulation results indicated that BBO/Complex is a competitive multidisciplinary optimization algorithm.

In the second part of Chapter 4, a Markov model was derived for BBO/Complex, and it was confirmed by a bi-subsystem sample problem. When the mutation was low – 0.001 or 0.01 – the optimal vector dominated the population with a probability of around 95% and 60% respectively. But with a high mutation rate of 0.1, the probability of obtaining the optimal population vector was only around 2.7%. Although the population probabilities were different with different mutation rates, the theoretical results calculated by the Markov model matched the simulations well, thus confirming the Markov model. According to our results, the computational requirements of the Markov model can be much less than those of simulations for small problems. Markov models are useful for

predicting the performance of heuristic algorithms, and quantifying the performance of different components in a heuristic algorithm without relying on long simulation times. Markov models can thus be helpful for algorithm design and parameter tuning. But Markov models also have a disadvantage. The computational effort can be very high for large problems. For a complex system with M subsystems, the total number of possible populations is

$$T = \prod_{i=1}^{M} \binom{n_i + N_i - 1}{N_i} \tag{5.1}$$

n_i : cardinality of search space in subsystem i
N_i : population size of subsystem i

Based on this equation, the total number of possible populations in our small sample system was 1,225. When we have a larger population size or a non-binary problem, this number will increase to an extremely large number that will result in a large transition matrix that cannot be handled with current computational resources.

5.2 Future Work

In this dissertation, we introduced three topics: efficiency tests and convergence analyses of heuristic algorithms, BBO for combinatorial problems, and BBO for complex systems. In the next step of our research, we will continue in these three directions.

First, we discussed the efficiency tests and convergence analyses for heuristic algorithms in this dissertation. But our conclusions were based on the probability

calculation after one generation. In the next step of our research, we can derive the percentage of the occurrence of optimal results based on a Markov model of the algorithm or a dynamic system model for each of the modified versions of BBO. We can use these models to analyze the performance of new variations of BBO.

Second, combinatorial problems are challenging benchmarks for heuristic algorithms. In order to improve the performance of BBO, we introduced new migration methods, local optimization methods, population initialization methods, and greedy methods. In future research, new techniques will be introduced to create hybrid BBOs dedicated to combinatorial problems. We also want to extend our research to real world applications, such as vehicle routing problems. We also want to use other popular solution methods like GA and ACO to solve the same problem for comparison.

Third, future work for BBO/Complex can be extended in four directions: convergence speed, adaptation, computational efficiency, and advanced testing. Convergence speed is one of the primary concerns for heuristic algorithms. Parallel computation can be used to decrease convergence time by dividing a task into multiple subtasks and solving them in parallel. One of the classic parallel computation models is the master-slave model. The master is in charge of job assignment and global calculations. The slaves perform subtasks that are assigned by the master, and return the results to the master. This structure can be adapted to BBO/Complex by viewing the master as the system optimizer and each slave as a subsystem optimizer. Computation time can be decreased dramatically with this structure, especially for problems with a large number of subsystems.

The second direction for future research in BBO/Complex is adaptation. In BBO/Complex, we find a solution to a complex system with a combination of within-subsystem migration and cross-subsystem migration. But other types of migration could also be implemented. A proper migration method can significantly increase performance for different types of problems. So we can design a series of migration methods, like migration for complex systems with tight subsystem coupling, migration for complex systems with loose subsystem coupling, migration for complex systems with many design variables, etc. Then we can classify the migration methods according to their performances on various types of problems and create a BBO/Complex algorithm that adaptively chooses the most efficient migration methods according to the selected problem.

The third direction for future research involves the computational effort of Markov modeling. Because of the heavy computational burden mentioned above, Markov models are limited to problems with small population sizes and binary island structures, which do not capture the structure of real world problems. This limitation might be able to partially addressed by combining similar Markov model states into a single state [74].

The last direction for future research in the area of BBO/Complex is further testing. As mentioned in Chapter IV, complex systems typically contain multiple subsystems, multiple objectives, and multiple constraints. In this dissertation, a Markov model was developed for complex systems with multiple subsystems and multiple objectives. In future research, a Markov model can be developed for complex systems that also include multiple constraints.

REFERENCES

[1] D. Simon, “Biogeography-based optimization," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 6, pp. 702–713, 2008.

[2] D. Simon, *Evolutionary Optimization Algorithms*, John Wiley & Sons, 2013.

[3] P. Austin, *Cracking the Roulette Wheel: The System & Story of the CPA Who Cracked the Roulette Wheel*, CreateSpace Independent Publishing Platform, 2010.

[4] R. Rarick, D. Simon, F. Villaseca, and B. Vyakaranam, “Biogeography-based optimization and the solution of the power flow problem,” *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, pp. 1029–1034, 2009.

[5] P. Lozovyy, G. Thomas, and D. Simon, “Biogeography-based optimization for robot controller tuning,” in: *Computational Modeling and Simulation of Intellect: Current State and Future Perspectives* (B. Igelnik, editor), IGI Global, pp. 162–181, 2011.

[6] D. Du and D. Simon, “Biogeography-Based Optimization for Large Scale Combinatorial Problems,” in: *Efficiency and Scalability Methods for Computational Intellect* (B.Igelnik and J. Zurada, editors), Chapter 10, pp. 197–217, IGI Global, 2013.

[7] D. Simon, M. Ergezer, D. Du, and R. Rarick, “Markov models for biogeography-based optimization," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 41, no. 1, pp. 299–306, 2011.

[8] D. Simon, “A Dynamic System Model of Biogeography-Based Optimization,” *Applied Soft Computing*, vol. 11, no. 8, pp. 5652–5661, 2011.

[9] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multi- objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[10] T. Yamada and R. Nakano, "A genetic algorithm with multi-step crossover for job-shop scheduling problems," *IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pp. 146–151, 1995.

[11] A. Nemirovskii, "Several NP-hard problems arising in robust stability analysis." *Mathematics of Control, Signals and Systems*, vol. 6, no. 2, pp. 99–105, 1993.

[12] W. Guo, C. Huang, L. Wang, and Q. Wu, “Hybrid BBO and GA algorithms based on elite operation," *Journal of Information & Computational Science*, vol. 9, no. 11, pp. 2987–2995, 2012.

[13] D. Du, D. Simon, and M. Ergezer, “Biogeography-based optimization combined with evolutionary strategy and immigration refusal," *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, TX, pp. 1023–1028, October 2009.

[14] G. Wang, L. Guo, H. Duan, L. Liu, H. Wang, and M. Shao, “Path planning for uninhabited combat aerial vehicle using hybrid meta-heuristic DE/BBO algorithm," *Advanced Science, Engineering and Medicine*, vol. 4, no. 6, pp. 550–564, 2012.

[15] M. Sood and M. Kaur, “Shortest path finding in country using hybrid approach of BBO and BCO," *International Journal of Computer Applications*, vol. 40, no. 6, pp. 9–13, 2012.

[16] L. Goel, D. Gupta, and V. Panchal, “Hybrid bio-inspired techniques for land cover feature extraction: A remote sensing perspective," *Applied Soft Computing*, vol. 12, no. 2, pp. 832–849, 2012.

[17] G. Dantzig and J. Ramser, “The truck dispatching problem," *Management Science*, vol. 6, pp. 81–91, 1959.

[18] R. Bowerman, B. Hall, and P. Calamai. "A multi-objective optimization approach to urban school bus routing: Formulation and solution method," *Transportation Research Part A: Policy and Practice*, vol. 29, no. 2, pp. 107–123, 1995.

[19] L. Dong and C. Xiang. "Ant colony optimization for VRP and mail delivery problems," *IEEE International Conference on Industrial Informatics*, Singapore, pp. 1143–1148, 2006.

[20] G. Mathews, “On the partition of numbers," *London Mathematical Society*, vol. 28, pp. 486–490, 1897.

[21] R. Nauss, "The 0-1 knapsack problem with multiple choice constraints." *European Journal of Operational Research*, vol. 2, no. 2, pp. 125–131, 1978.

[22] M. Mitchell, *An introduction to genetic algorithms*, MIT Press, 1998.

[23] P. Boilot, “Electronic noses inter-comparison, data fusion and sensor selection in discrimination of standard fruit solutions," *Sensors and Actuators B: Chemical*, vol. 88, no. 1, pp. 80–88, 2003.

[24] M. Desrochers, “A classification scheme for vehicle routing and scheduling problems," *European Journal of Operational Research*, vol. 46, no. 3, pp. 322–332, 1990.

[25] A. Jaszkiewicz, “Genetic local search for multi-objective combinatorial optimization," *European Journal of Operational Research*, vol. 137, no. 1, pp. 50–71, 2002.

[26] H. Mo and L. Xu, “Biogeography migration algorithm for traveling salesman problem," in: *Advances in Swarm Intelligence* (Y. Tan, Y. Shi, and K. Tan editors), Springer, pp. 405–414, 2010.

[27] M. Ergezer and D. Simon, “Oppositional biogeography-based optimization for combinatorial problems," *IEEE Congress On Evolutionary Computation*, New Orleans, LA, pp. 1496–1503, 2011.

[28] D. Du and D. Simon, “Complex System Optimization Using Biogeography-Based Optimization,” *Mathematical Problems in Engineering*, vol. 2013, Article ID 456232, 17 pages, 2013.

[29] P. Cilliers, *Complexity and postmodernism: understanding complex systems,* Routledge, 1998.

[30] J. Allison, *Complex system optimization: A review of analytical target cascading, collaborative optimization, and other formulations*, M.S. Thesis, Mechanical Engineering Department, University of Michigan, Ann Arbor, MI, 2004.

[31] S. Bradley, A. Hax, and T. Magnanti, *Applied Mathematical Programming*, Addison Wesley, 1977.

[32] L. Vicente and P. Calamai, “Bilevel and multilevel programming: A bibliography review," *Journal of Global Optimization*, vol. 5, no. 3, pp. 291–306, 1994.

[33] M. Masmoudi and D. Auroux, “The state of the art in collaborative design," in: *Recent Trends in Aerospace Design and Optimization* (B. Uthup, S. Koruthu, R. Sharma, and P. Priyadarshi, editors), Tata McGraw Hill, pp. 411–425, 2005.

[34] L. Leblanc and D. Boyee, “A bilevel programming algorithm for exact solution of the network design problem with user-optimal flows," *Transportation Research*, vol. 20, vol. B, pp. 259–265, 1986.

[35] J. Bard, “Coordination of a multidivisional organization through two levels of management," *OMEGA*, vol. 11, pp. 457–468, 1983.

[36] B. Hobbs and S. Nelson, “A nonlinear bilevel model for analysis of electric utility demand-side planning issues," *Annals of Operations Research*, vol. 34, pp. 255–274, 1992.

[37] W. Walker, "A heuristic adjacent extreme point algorithm for the fixed charge problem," *Management Science*, vol. 22, no. 5, pp. 587–596, 1976.

[38] R. Braun, P. Gage, and I. Kroo, “Implementation and performance issues in collaborative optimization,” *AIAA/NASA/ISSMO, Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, Washington, 1996.

[39] J. Martins and A. Lambe, “Multidisciplinary design optimization: A Survey of architectures,” *AIAA Journal*, vol. 59, no. 9, pp. 2049–2075, 2013.

[40] E. Cramer, J. Dennis, P. Frank, R. Lewis, and G. Shubin, “Problem formulation for multidisciplinary optimization,” *SIAM Journal of Optimization*, vol. 4, no. 4, pp. 754–776, 1994.

[41] S. Kodiyalam and J. Sobieszczanski-Sobieski, “Multidisciplinary design optimization - some formal methods, framework requirements, and application to vehicle design,” *International Journal for Vehicle Design*, vol. 25, no. 1 and 2, pp. 3–22, 2000.

[42] T. Zang and L. Green, “Multidisciplinary design optimization techniques: Implications and opportunities for fluid dynamics research,” *30th AIAA Fluid Dynamics Conference*, Norfolk, Virginia, 1999.

[43] G. Reinelt, “TSPLib - A traveling salesman problem library," *ORSA Journal On Computing*, vol. 3, pp. 376–384, 1991.

[44] T. CoverHart, “Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[45] G. Syswerda, “Schedule optimization using genetic algorithms," in: *Handbook of Genetic Algorithms* (L. Davis editor), Van Nostrand Reinhold, pp. 332–349, 2011.

[46] H. Lee, A. Battle, R. Raina, and A. Y. Ng. "Efficient sparse coding algorithms," *Advances in neural information processing systems*, vol. 19, pp. 801–808, 2007.

[47] B. Fox and M. McMahon, “Genetic operators for sequencing problems," in: *Foundations of Genetic Algorithms 1* (G. Rawlins editor), Morgan Kaufmann, pp. 284–300, 1991.

[48] I. Oliver, D. Smith, and J. Holland, “A study of permutation crossover operators on the traveling salesman problem," S*econd International Conference on Genetic Algorithm and Their Application*, Hillsdale, NJ, pp. 224– 230, 1987.

[49] G. Tao and Z. Michalewicz, “Inver-over operator for the TSP," *Parallel Problem Solving From Nature V*, pp. 803–812, 1998.

[50] D. Johnson and L. McGeoch, “The traveling salesman problem: A case study in local optimization," in: *Local Search in Combinatorial Optimization* (E. Aarts, J. Lenstra editors), John Wiley Sons, pp. 215–310, 1997.

[51] G. Gutin, A. Yeo, and A. Zverovich, “Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP," *Discrete Applied Mathematics*, vol. 117, pp. 81–86, 2002.

[52] P. Poon and J. Carter, “Genetic algorithm crossover operators for ordering applications," *Computers Operations Research*, vol. 22, no. 1, pp. 135–147, 1995.

[53] M. Dorigo and L. Gambardella, “Ant colonies for the traveling salesman problem," *Biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[54] E. Aarts, “Simulated annealing: An introduction," *Statistica Neerlandica*, vol. 43, no. 1, pp. 31–52, 1989.

[55] J. Abell and D. Du, “A framework for multiobjective, biogeography-based optimization of complex system families," *AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, TX, September 2010.

[56] A. Bhattacharya and P. Chattopadhyay, “Application of biogeography-based optimization for solving multi-objective economic emission load dispatch problems,” *Electric Power Components and Systems*, vol. 38, no. 3, pp. 340–365, 2010.

[57] P. Roy, S. Ghoshal, and S. Thakur, “Biogeography based optimization technique applied to multi-constraints economic load dispatch problems,” *Transmission and Distribution Conference and Exposition: Asia and Pacific*, Seoul, South Korea, 2009.

[58] J. Lin, "Multiple-objective problems: Pareto-optimal solutions by method of proper equality constraints." *IEEE Transactions on Automatic Control*, vol. 21, no. 5, pp. 641–650, 1976.

[59] D. Goldberg, *Genetic algorithm in search, optimization, and machine learning*, Addison-Wesley, 1989.

[60] N. Srinivas and K. Deb, “Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms,” *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 3, pp. 221–248, 1994.

[61] M. Fonseca and P. Fleming, "Genetic algorithms for multiobjective optimization: formulation, discussion and generalization." *International Conference on Genetic Algorithms*, Urbana-Champaign, IL, pp. 416–423, 1993.

[62] R. Hathaway and J. Bezdek, “Fuzzy c-means clustering of incomplete data,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 31, no. 5, pp. 735–744, 2001.

[63] S. Kodiyalam, *Evaluation of methods for multidisciplinary design optimization (MDO), Phase I*, National Aeronautics and Space Administration, NASA CR-1998-208716, Langley Research Center, 1998.

[64] X. Chen, B. Li, and Y. Lin, "Multidisciplinary design optimization with a new effective method," *Chinese Journal Of Mechanical Engineering*, vol. 23, no. 4, 2010.

[65] M. Xiao, L. Gao, H. Qiu , X. Shao, and X. Chu, "An Approach Based on Enhanced Collaborative Optimization and Kriging Approximation in Multidisciplinary Design Optimization," *Advanced Materials Research*, vol. 118, pp. 399–403, 2010.

[66] K. Jamuna and K. Swarup, "Multi-objective biogeography based optimization for optimal PMU placement," *Applied Soft Computing*, vol. 12, no. 5, pp. 1503–1510, 2012.

[67] P. Roy and D. Mandal, "Quasi-oppositional biogeography-based optimization for multi-objective optimal power flow," *Electric Power Components and Systems*, vol. 40, no. 2, pp. 236–256, 2011.

[68] J. Suzuki, "A Markov chain analysis on simple genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 25, no. 4, pp. 655–659, 1995.

[69] J. Suzuki, "A further result on the Markov chain model of genetic algorithms and its application to a simulated annealing-like strategy," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 28, no. 1, pp. 95–102, 1998.

[70] A. Wright and Y. Zhao, "Markov chain models of genetic algorithms," *Genetic and Evolutionary Computation Conference*, vol. 1, pp. 734-741, 1999.

[71] C. Grinstead and J. Snell, *Introduction to probability*. Providence, RI: American Mathematical Soc., 1998.

[72] N. Beaulieu, "On the generalized multinomial distribution, optimal multinomial detectors, and generalized weighted partial decision detectors," *IEEE Transactions on Communications*, vol. 39, no. 2, pp. 193–194, 1991.

[73] G. Rudolph, "Convergence analysis of canonical genetic algorithms," *IEEE Transactions on Neural Networks*, vol. 5, no. 1, pp. 96–101, 1994.

[74] C. Reeves and J. Rowe, *Genetic Algorithms: Principles and Perspectives*, Springer, 2002.

[75] L. Padula, N. Alexandrov, and L. Green, "MDO test suite at NASA Langley research center," *AIAA/NASA/ISSMO, Symposium on Multidisciplinary Analysis and Optimization*, Bellevue, Washington, 1996.

[76] A. Morgan, A. Sommese, and L. Watson, "Mathematical reduction of a heart dipole model," *Journal of Computational and Applied Mathematics*, vol. 27, no. 3, pp. 407–410, 1989.

[77] N. Tedford, J. Martins, "Benchmarking multidisciplinary design optimization algorithms," *Optimization and Engineering*, vol. 11, no. 1, pp. 159–183, 2010.

APPENDICES

APPENDIX A: DAWEI DU'S PUBLICATIONS

[1] D. Du, D. Simon, and M. Ergezer, "Biogeography-based optimization combined with evolutionary strategy and immigration refusal," *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, TX, pp. 1023–1028, 2009.

[2] M. Ergezer, D. Simon, and D. Du, "Oppositional biogeography-based optimization," *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, pp. 1035–1040, October 2009.

[3] D. Simon, M. Ergezer, and D. Du, "Population distributions in biogeography- based optimization algorithms with elitism," *IEEE Conference on Systems, Man, and Cybernetics*, San Antonio, Texas, pp. 1017–1022, 2009.

[4] J. Abell and D. Du, "A framework for multiobjective, biogeography-based optimization of complex system families," *AIAA/ISSMO Multidisciplinary Analysis Optimization Conference*, Fort Worth, Texas, 2010.

[5] D. Simon, M. Ergezer, D. Du, and R. Rarick, "Markov models for biogeography-based optimization," I*EEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 41, no. 1, pp. 299–306, 2011.

[6] D. Simon, R. Rarick, M. Ergezer, and D. Du, “Analytical and numerical comparisons of biogeography-based optimization and genetic algorithms," *Information Sciences*, vol. 181, no. 7, pp. 1224–1248, 2011.

[7] D. Du and D. Simon, “Biogeography-Based Optimization for Large Scale Combinatorial Problems," in: *Efficiency and Scalability Methods for Computational Intellect* (B. Igelnik, J. Zurada editors), IGI Global, Chapter 10, pp. 197–217, IGI Global, 2013.

[8] D. Du and D. Simon, “Complex System Optimization Using Biogeography-Based Optimization,” *Mathematical Problems in Engineering*, vol. 2013, Article ID 456232, 17 pages, 2013.

APPENDIX B: BENCHMARK PROBLEMS FOR BBO/COMPLEX

This appendix gives details about the benchmark problems used in this dissertation.

Speed Reducer

The speed reducer problem is a gear box design problem [62], [64], [75]. The objective is to minimize the gear box weight, and the von Mises stresses for shaft 1 and 2. This problem contains 3 objectives, 11 constraints, and 7 design variables. This problem is defined as follows.

$$\min\ F_1 = 0.7854x_1x_2^{\,2}(3.3333x_3^{\,2} + 14.9334x_3 - 43.0934) - 1.5079x_1(x_6^{\,2} + x_7^{\,2}) + 7.477(x_6^{\,3} + x_7^{\,3}) + 0.7854(x_4x_6^{\,2} + x_5x_7^{\,2})$$

$$\min\ F_2 = \sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2 + 1.69\times 10^7}$$

$$\min\ F_3 = \sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2 + 1.575\times 10^8}$$

such that

$$g_1 = \frac{27}{x_1x_2^{\,2}x_3} - 1 \le 0$$

$$g_2 = \frac{397.5}{x_1x_2^{\,2}x_3^{\,2}} - 1 \le 0$$

$$g_3 = \frac{1.93x_4^{\,3}}{x_2x_3x_6^{\,4}} - 1 \le 0$$

$$g_4 = \frac{1.93x_5^{\,3}}{x_2x_3x_7^{\,4}} - 1 \le 0$$

$$g_5 = \frac{\sqrt{\left(\frac{745x_4}{x_2x_3}\right) + 1.69\times10^7}}{0.1x_6^{\,3}} - 1100 \le 0$$

$$g_6 = \frac{\sqrt{\left(\frac{745x_5}{x_2x_3}\right) + 1.575\times10^8}}{0.1x_6^{\,3}} - 850 \le 0$$

$$g_7 = x_2x_3 - 40 \le 0$$

$$g_8 = \frac{x_1}{x_2} - 12 \le 0$$

$$g_9 = \frac{-x_1}{x_2} + 4 \le 0$$

$$g_{10} = \frac{1.5x_6 + 1.9}{x_4} - 1 \le 0$$

$$g_{11} = \frac{1.1x_7 + 1.9}{x_5} - 1 \le 0$$

The objectives, decision variables, and constraints are defined as follows.

F_1 : overall weight of gearbox
F_2 : von Mises stress for shaft 1
F_3 : von Mises stress for shaft 2

x_1 : gear face width
x_2 : tooth module
x_3 : number of teeth of pinion
x_4 : distance between bearing 1
x_5 : distance between bearing 2
x_6 : diameter of shaft 1
x_7 : diameter of shaft 2

g_1 : bending stress of gear tooth
g_2 : contact stress of gear tooth
g_3 : transverse deflection of shaft 1
g_4 : transverse deflection of shaft 2
g_5 : stress in shaft 1
g_6 : stress in shaft 2
$g_7 - g_{11}$: dimension requirement for shafts

Power Converter

The power converter problem [62], [75] consists of two subsystems – the electrical subsystem and the loss subsystem. It has 6 design variables, 8 state variables, 2 objectives, and 4 constraints. The system is described as follows.

$$\min\ F_1 = \left| 0.78\times 10^4 x_1^{\,2}\left(6x_6 + \frac{\pi x_1}{2}\right)\right| + \left|6.747\times 10^4 x_1 x_2 x_3\right|$$

$$\min\ F_2 = \left|25x_5\right| + \left|\frac{5\times 10^2(1-y_2)}{88y_2}\right|$$

such that

$$g_1 = \frac{2x_2(x_2 - 2\times 10^{-3} - x_2 x_3)}{0.4} \geq 0$$

$$g_2 = \frac{5\times10^2 - \frac{5.65(1-y_3)}{10^5 x_4}\frac{0.3\times10^{-4}}{x_5}}{5} \geq 0$$

$$g_3 = 0.3 - \frac{x_4\left(100 + \frac{5.65(1-y_4)0.5}{10^5 x_4}\right)}{x_2 y_6} \geq 0$$

$$g_4 = x_4 - \frac{28.25(1-y_4)}{10^7} \geq 0$$

State variables:

$$\mathrm{y}_1 = \left|0.78\times10^4 x_1{}^2\left(6x_6 + \frac{\pi x_1}{2}\right)\right| + \left|6.747\times10^4 x_1 x_2 x_3\right| + \left|25x_5\right| + \left|\frac{5\times10^2(1-y_2)}{88y_2}\right|$$

$$y_2 = \frac{500}{y_3\frac{3.25\times10^2}{32}}$$

$$y_3 = \frac{500}{y_2\frac{3.25\times10^2}{32}}$$

$$y_4 = \frac{500}{y_2\frac{4.25\times10^2}{32}}$$

$$y_5 = \frac{7.6x_1x_2 1.724\times10^{-8}}{x_3}$$

$$y_6 = x_1{}^2$$

$$y_7 = \frac{\pi x_1}{2}$$

$$y_8 = \frac{5.65(1+y_3)}{y_6 x_2 10^5}$$

The objectives, decision variables, states, and constraints are defined as follows.

F_1 : weight of primary winding
F_2 : weight of secondary winding

x_1 : core center leg width
x_2 : turns
x_3 : copper size
x_4 : inductance
x_5 : capacitance
x_6 : core window width

y_1 : component weight
y_2 : circuit efficiency
y_3 : duty cycle
y_4 : minimum duty cycle
y_5 : inductor resistance
y_6 : core cross-sectional area
y_7 : magnetic path length
y_8 : inductor value

g_1 : fill window constraint
g_2 : ripple specification
g_3 : core saturation
g_4 : minimum inductor size

Heart Dipole

The heart dipole problem [62], [75], [76] is based on the electrolytic determination of the dipole moment in the heart. This problem contains 2 objectives, 5

constraints, and 6 design variables. This problem was modified from its original formulation in order to be testable with MDO algorithms. Therefore, although the problem is a common MDO benchmark, the objectives do not have any physical meaning. The problem is defined as follows.

$$\min\ F_1 = x_1((1-x_2)^2 - x_3^{\,2}) - 2x_1(1-x_2)x_3 + (1-x_1)(x_2^{\,2} - x_4^{\,2}) - 2(1-x_1)x_2x_4 - 1 + x_1((1-x_2)^2 - x_3^{\,2}) + 2x_1(1-x_2)x_3 + (1-x_1)(x_2^{\,2} - x_4^{\,2}) + 2(1-x_1)x_2x_4 - 1$$

$$\min\ F_2 = x_1(1-x_2)((1-x_2)^2 - 3x_3^{\,2}) + x_1x_3(x_3^{\,2} - 3(1-x_2^{\,2})) + (1-x_1)x_2(x_2^{\,2} - 3x_4^{\,2}) + (1-x_1)x_4(x_4^{\,2} - 3x_2^{\,2}) - 1 + x_1(1-x_2)((1-x_2)^2 - 3x_3^{\,2}) - x_1x_3(x_3^{\,2} - 3(1-x_2)^2) + (1-x_1)x_2(x_2^{\,2} - 3x_4^{\,2}) - (1-x_1)x_4(x_4^{\,2} - x_2^{\,2}) - 1$$

such that

$$g_1 = \left| x_3x_1 + x_4(1-x_1) - x_5(1-x_2) - x_6x_2 - 1 \right| < 0.1$$

$$g_2 = \left| x_5x_1 + x_6(1-x_1) + x_3(1-x_2) + x_4x_2 - 1 \right| < 0.1$$

$$g_3 = x_1((1-x_2)^2 - x_3^{\,2}) - 2x_1(1-x_2)x_3 + (1-x_1)(x_2^{\,2} - x_4^{\,2}) - 2(1-x_1)x_2x_4 - 1 > 0$$

$$g_4 = x_1((1-x_2)^2 - x_3^{\,2}) + 2x_1(1-x_2)x_3 + (1-x_1)(x_2^{\,2} - x_4^{\,2}) + 2(1-x_1)x_2x_4 - 1 > 0$$

$$g_5 = x_1(1-x_2)((1-x_2)^2 - 3x_3^{\,2}) + x_1x_3(x_3^{\,2} - 3(1-x_2)^2) + (1-x_1)x_2(x_2^{\,2} - 3x_4^{\,2}) + (1-x_1)x_4(x_4^{\,2} - 3x_2^{\,2}) - 1 > 0$$

$$g_6 = x_1(1-x_2)((1-x_2)^2 - 3x_3^{\,2}) - x_1x_3(x_3^{\,2} - 3(1-x_2)^2) + (1-x_1)x_2(x_2^{\,2} - 3x_4^{\,2}) - (1-x_1)x_4(x_4^{\,2} - 3x_2^{\,2}) - 1 > 0$$

The objectives, decision variables, and constraints are defined as follows.

F_1 : sum of g_3 and g_4
F_2 : sum of g_5 and g_6

x_1 : magnitude of dipole 1 on x-axis
x_2 : magnitude of dipole 2 on x-axis
x_3 : magnitude of dipole 1 on y-axis
x_4 : magnitude of dipole 2 on y-axis
x_5 : coordinate of dipole 1 on x-axis
x_6 : coordinate of dipole 2 on x-axis
x_7 : coordinate of dipole 1 on y-axis
x_8 : coordinate of dipole 2 on y-axis

$g_1 - g_6$: predefined constraints to determine the magnitude, directions, and locations of two dipoles.

Propane Combustion

The propane combustion problem is a chemical equilibrium problem [62], [75], [77]. This problem contains 3 objectives, 4 constraints, and 11 design variables. This problem is described as follows.

$$\min F_1 = 2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} - 10$$

$$\min F_2 = \sqrt{x_2 x_4} - x_6 \sqrt{\frac{40x_1}{x_{11}}},\ x_{11} = \sum_{i=1}^{i=10} x_i$$

$$\min F_3 = \sqrt{x_1 x_2} - x_7 \sqrt{\frac{40x_4}{x_{11}}} + x_1 \sqrt{x_3} - x_4 x_9 \sqrt{\frac{40}{x_{11}}}$$

such that

$$g_1 = 2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} - 10 > 0$$

$$g_2 = \sqrt{x_2 x_4} - x_6\sqrt{\frac{40x_1}{x_{11}}} > 0$$

$$g_3 = \sqrt{x_1 x_2} - x_7\sqrt{\frac{40x_4}{x_{11}}} > 0$$

$$g_4 = x_1\sqrt{x_3} - x_4 x_9\sqrt{\frac{40}{x_{11}}} > 0$$

The objectives, decision variables, and constraints are defined as follows.

F_1 : first product of combustion
F_2 : second product of combustion
F_3 : sum of third and fourth product of combustion

$x_1 - x_{10}$: number of moles of each product formed
for each mole of propane burned
x_{11} : sum of x_1 to x_{10}

g_1 : first product of combustion
g_2 : second product of combustion
g_3 : third product of combustion
g_4 : fourth product of combustion